PREDICTING MOVIE RATINGS AND RECOMMENDER SYSTEMS

Arkadiusz Paterek

Predicting movie ratings and recommender systems

Arkadiusz Paterek

Jun 19, 2012

Contents

Abstract

This monograph describes author's large experimental work on one machine learning task – prediction of movie ratings in the Netflix Prize dataset. The main objective of the experiments was to obtain maximally accurate prediction, as evaluated by hold-out RMSE, but also important was the perspective of applying the developed methods in recommender systems. The publication has two goals: summarizing the understanding of the subject due to the published work of many people on the same task, and presenting some novel insights. Reaching a good understanding of one task and one dataset gives hope to generalize on other prediction tasks, as similar challenges recur in analyses of any datasets.

The idea of collaborative filtering is to make use of relations between tasks (users in our data), and between task attributes (items in our data). Collaborative filtering methods are used in recommender systems to calculate personalized recommendations, or in other words, to identify items preferred by a particular user. To realize that goal, a good intermediate task is prediction of user ratings, and the most accurate models for this task are based on dimensionality reduction, describing each item by a small number of variables, which can be seen as automatically learned analogues of movie genres, and a small number of variables describes each user's taste. One the most accurate models, regularized SVD, was analyzed more closely, and the assumptions of that model, such as the single-variable output, combining hidden variables by multiplication, and using Gaussian priors, were critically examined. In addition, an interpretation of the learned features by naming new movie genres has been proposed.

To learn the parameters in the developed models the best predictive accuracy was obtained by using different degrees of approximation of the Bayesian approach, from MCMC and Variational Bayes, to neural-networks-like simplifications. When identifying the model, that is, while approaching the unknown probabilistic model that generated the data, good engineering practice was maintaining a blend of an ensemble of many accurate, but varied methods. Blends of large ensembles also gave the best reached accuracy, indicating that, despite the large combined effort of many people, the process of model identification for the analyzed data remained largely unfinished, which is probably an unavoidable situation in an analysis of real-life datasets.

The work is complemented by giving heuristics adapting rating prediction to generate lists of recommendations, heuristics for cold-start situations, and descriptions of two SVD-based recommender systems.

Acknowledgements

To John Tomfohr for sharing details about his very accurate Bayesian model. The knowledge of this model influenced my experiments and this book the most, and it helped me better understand the fundamentals of what is important in predictive modelling.

To Piotr Pokarowski. A large part of the methodology of statistical data analysis I use I learned on his course.

To Andriy Mnih for answering my questions.

To Morgan Stanley PDT group, where with John we gave a talk about our Netflix Prize solutions, for the hint about "Bayesing users" more heavily.

To all people that helped me on my journey to understand machine learning and science in general.

To friends, without whose support this book would never have been finished.

You know exactly what we have to do -
To give them something with a kick in it,
So hurry up and make a decent brew.
Don't leave it till tomorrow, stick at it -
Today will pass you by before you know.
You've got to grab your chance, or else it's gone,
It doesn't come round twice, so don't be slow,
And once you've taken it, don't let it go;
That's the only way to get things done.

Faust, J.W.Goethe [1]

1 Introduction

One of the biggest needs of our times is to make use of data, which enormous amount is gathered in digital form. It would be good if the level of development and understanding of the science, craft and art of data analysis matched the importance of that domain. It is no exaggeration to say that the collection, understanding, and use of data are large parts of every area of science (for applied sciences are the foundation, but are also needed in mathematical, abstract sciences), that is why even small developments in the field of data analysis can have a large impact, be useful across many domains.

A large fraction of emerging tasks of data analysis, for which there is a need of precise defining and solving, are prediction tasks, that is reasoning about the unknown, unobserved part of reality on the basis of the observed part of reality in the form of gathered data. This work will be devoted to one chosen prediction task, with a single dataset. It should be emphasised that there is no known optimal approach to the analysis of real-life datasets. The laws of physics discovered based on observed data (both passive gathering and controlled experiments) describe only small parts of reality, and do it imprecisely. Similarly so, in prediction tasks, like the one described in this work, one has to be prepared that the theory and practice developed so far do not describe accurately all phenomena encountered, do not show right away the best approaches to prediction for analyzed real-life data. We expect that it is necessary to take an approach to a large extent phenomenological, guided by experiments and discovery.

Because prediction is so prevalent and useful, there is a need to understand this field thoroughly. The overall concept for the research described in this work is to examine one prediction task really well (typically, little time is devoted to most prediction tasks, and examination is shallow). The conditions needed to perform a thorough and useful examination are: having a large, good quality dataset, a fixed prediction task and evaluation criterion for solutions, and involvement of many experts for a long time. Such an occasion was the Netflix Prize contest, where a dataset of over 100 million ratings was released, and a task of predicting unknown ratings was proposed, with accuracy evaluated by RMSE (root mean squared error). During several years since making the data available many professionals in various fields faced the same task (the contest website summarized that over 5000 teams submitted their solutions), and a large part of the developed methods was published. This work, besides presenting novel material, attempts to summarize the published efforts of many people put into investigating the Netflix task. The work de-

[1] Transl. of *Faust* by John R. Williams

scribes the most accurate methods found, and the methods resulting in the best combined accuracy of the ensemble, but important for us will be also the context of using the methods in recommender systems and opportunities to generalize on other prediction tasks. We are also interested in optimizations on the meta-level: minimizing analyst's time and effort, improving the general engineering process of approaching the possibly most accurate prediction, model identification, how to quickly identify all important effects, finding convenient ways to simplify and automate.

Besides the potential to develop the field of prediction, a large advantage of the chosen task of rating prediction is usefulness of accurate solutions. Ratings are one of the best known indicators of user's preferences for items of any kind. Ratings have the advantage, that they allow to indicate both positive and negative user preference for an item, and that it is explicit feedback, where the collected preferences are close to the real preferences, to what the user really thinks. The alternatives are implicit (passively gathered) types of feedback, like clickstream, which are easier to collect in large amounts, but their common drawbacks are: their meaning is not always clear, they introduce privacy anxieties, and, when used in search and recommender systems, they are often prone to loopback effects, reinforcing regions of lower data quality. In short, explicit feedback data is easier to use correctly. Prediction of user preferences is a foundation of recommender systems, which are gaining much popularity lately. Recommender systems are becoming, next to search, a basic methodology of information retrieval – a domain, which overall objective is helping to save user's time needed to reach information that interests him. Recommender systems realize the possibility of presenting relevant items without specifying a query by the user, but based on his past activity. It is likely that in the future the most efficient systems of information retrieval will be hybrids of personalized recommendations and search (two views, essentially the same, are personalized search and recommendations filtered by search). To calculate rating-based personalized recommendations a good intermediate task is prediction of unknown user ratings for items. In this work the prediction accuracy is evaluated by RMSE (root mean squared error) on a held-out test set, and we focus on optimizing this criterion.

Let's briefly introduce to the methodology of prediction used in this work. Various approaches to capturing uncertainty in data were proposed in the past, but the most appropriate, most fundamental approach is based on probability theory and Bayes law, where the collected data is used to calculate the posterior probability of model parameters:

$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

where D is the data, and θ are the parameters of the model. We can note that (apart from trivial models) no matter how much data was collected, there is always uncertainty left about what is the exact form of the real model, about which we can say that it generated the data. A large part of the difficulty of prediction tasks is to identify the unknown model accurately enough. The identified and learned model is used in the so-called decision-theoretic approach to prediction, that is the predicted value is chosen so as to minimize the expected loss with respect to the posterior distribution of model parameters. This approach can be simplified by attempting to minimize the expected posterior loss directly, skipping the step of defining a probabilistic model. Approximations of Bayesian inference and of the decision-theoretic approach create a range of prediction methods, which are the foundation of a family of related disciplines: machine learning, statistical data analysis, predictive modelling, predictive analytics, data mining, knowledge discovery in databases, pattern recognition, artificial intelligence, and others. Efficient simplifications of the Bayesian approach are sometimes inspired by biology, as the methods of neural networks, or can be inspired by physics, as the model of Boltzmann machines, the method of Gibbs sampling, or optimization methods by simulated annealing. Prediction is

an art to a large degree experience-based, makes use of insights recurring in many different tasks and datasets, like the fact that the most observed distribution in nature is Gaussian distribution. Identification of the probabilistic model, choice of assumptions about the distributions, outliers, measurement error, missing values, are sufficiently difficult for data directly observed. The more difficult is to choose all assumptions for models with layers of hidden (latent) variables, such as the models being the core of this work, containing hidden variables representing user preferences and item features.

The methods solving the Netflix's task of rating prediction belong to the family of collaborative filtering methods. The concept of collaborative filtering evolved from the initial idea of filtering mailing lists by multiple moderators, hand-chosen by every user, to the current query-less model-based approaches, where all gathered ratings made by all users for all items influence in some way every predicted rating (and each user's list of recommendations). The prediction problem chosen here can be seen as predicting missing values in a sparse matrix. There is also a multitask learning view, where each of the related tasks is to predict ratings of one user.

Collaborative filtering is not only about predicting user preferences in recommender systems, there are applications in other domains, but we focus here on the most common application of predicting ratings. The idea behind collaborative filtering algorithms is that when users with similar taste rate similar items, the ratings will be close together. It is intuitively obvious (we can say that here manifest themselves pattern detection capabilities of human brain), that movies can be classified into genres, and there are groups of users who like or not a given genre. It turned out that this intuition of classifying movies into genres is reflected in the most accurate methods for the task of movie rating prediction, which include dimensionality reduction. For example, variations of matrix factorization [Fun06] automatically describe each item by a small number of parameters (30-200), called item features (for movies, you can see them as automatically learned genres), and a similarly small number of parameters defining user preferences for features. In addition to including a layer of dimensionality reduction, which explained most of the variability of ratings, also important was modelling on multiple scales [Bel07a]: adding a layer with few parameters, called global effects, and a highly parameterized layer that models direct item-item relationships.

In prediction often important is the aspect of making forecasts and the variability of the model over time. How important is modelling time variability in a recommender system, depends on the type of data. Because of the chosen task of predicting preferences for movies, here we are mostly interested in static aspects of prediction. For this type of data user preferences typically do not change much over the years (it is different for some other types of items, for example, for news articles, which rapidly cease to be interesting for users). Nevertheless capturing a certain degree of the time variability of the model improved accuracy of rating prediction. Particularly significant was modelling a single-day user bias.

To sum up the developments for the Netflix task, as the result of efforts of many people working on the task, methods were developed with the accuracy, presumably, close to best possible. Accurate prediction of ratings is (with some adaptations) a good intermediate goal for calculating lists of personalized recommendations. In addition, a side result of accurate algorithms predicting ratings are good quality item-item and user-user similarities, which are useful in various applications.

In this work I attempt to thoroughly review the topic of obtaining accurate prediction of movie ratings with collaborative filtering methods. We can think of the selection of presented methods as being filtered by what worked well on the Netflix task, and what did not. The presentation of the topic is based both on the large body of published work of many authors working independently on the same task, and is supported by own large

experimental work, which resulted also in several improvements of methods and novel insights.

I begin with introducing the domain of prediction in chapter 2. The basic approach of probabilistic modelling is outlined, the use of data for inference through Bayes' rule, and the understanding of prediction in the framework of decision theory. I describe linear regression, the most popular method of prediction, I discuss the model selection problem, and reflect on when accurate prediction is really needed.

Chapter 3 introduces the chosen task and dataset. As for us important is the perspective of actual application to calculate personalized recommendations, I begin by summarizing major issues to solve when developing recommender systems, and how rating prediction relates to generating lists of recommendations. Next I discuss the concept of collaborative filtering methods as an approach to rating prediction, and generally to filter items for personalized recommendations. Next the dataset is shortly described, the evaluation criteria used throughout the work, and the first summarization of the data by plots and tables.

Chapter 4 contains the main part of the work, a comprehensive overview of rating prediction methods. The description of methods is accompanied by results of author's implementations, and by listing the experimental results published by other authors. I begin with a discussion of the simplest models: only the global mean, only biases, and one-feature regularized singular value decomposition with biases. Simple models illustrate the emerging issues, common simplifications and choices to make in all methods in the general approach to prediction taken in this work. Experiments with nonparametric modelling were presented, that to some degree justify the form of the model of regularized SVD, and to some degree dispute it. I discuss how to model the output variable, and how to use the predictions to calculate recommendations. Next in section 4.3 variants of regularized SVD with multiple features are extensively described. Different ways to learn the models are discussed, with approximate Bayesian methods and also with neural networks-like approaches with cost function minimization. The resulting features of the SVD were visualized for exemplary movies. SVD features define certain notions that one can try to name. I attempted to interpret the first six features, that explain most of the variance in data, as a set of six pairs of new, experimental genres, much more informative from the viewpoint of rating prediction, in comparison to the standard set of genres. Next discussed are: improving estimation of user features by using implicit information, different ways of using time data, and the matrix norm regularization view. The following sections 4.4, 4.5, 4.6 discuss efficient approaches to collaborative filtering other than matrix factorization: restricted Boltzmann machines, KNN-related methods, kernel methods, and other methods. In section 4.7 I discuss the use of external item metadata. Metadata did not improve accuracy of methods on the Netflix Prize dataset, but it may be useful in recommender systems to overcome cold-start problem by content-based prediction of parameters. Section 4.8 overviews different ways of combining models: stacking methods on residuals of one another, building integrated models, and blending independently learned methods. Best predictive accuracy was obtained by blending of many accurate, but different methods. Maintaining a linearly blended ensemble was also a convenient framework that allowed to evaluate the gradually implemented methods, and that was helpful in exploring the space of possible models.

Chapter 5 summarizes the experimental results and presents the final ensemble of methods in a sequence of feature selection, which allows to assess importance of individual methods. The first few most contributing methods are the individual methods and combinations of the following: matrix factorization, RBM, kernel methods, K-NN, combined with global effects, variability of models in time, and using the structure of missing data.

Chapter 6 describes the construction of SVD-based recommender systems in two (pri-

vate) side-projects, and also the use of SVD results for clustering and visualizations, used in applications that help in discovering similar items.

Finally, chapter 7 summarizes the work with conclusions about collaborative filtering prediction and with general insights about prediction, drawn from the extensive analysis of the large Netflix dataset.

The aim of a scientific work is to introduce innovations. The previous publication of the author [Pat07] introduced techniques as adding biases to the regularized SVD (proposed also in [Tak07a, Tak07b]), postprocessing SVD with Kernel Ridge Regression, and introduced the methods NSVD1, NSVD2. In this work the novel material is:

- a simple and largely improving accuracy modification of the covariance matrix in KRR by a time-dependent term (and a similar term improving item-item similarities in K-NN methods),
- a directed version of RBM with two sets of weights,
- several other minor accuracy improvements,
- experiments with nonparametric modelling, verifying the form of priors for regularized SVD and verifying the choice of multiplication out of possible two-parameter functions – the main conclusion here was that the common choice of Gaussian priors for parameters of matrix factorization models is inaccurate.

In addition, there are insights for recommender systems and related applications, how to adapt rating prediction to create lists of personalized recommendations, how to heuristically adjust the rating prediction for the prediction variance and for the missing data structure, and how to ensure diversity on a recommendation list. Heuristics are proposed for cold-start situations in recommender systems: content-based prediction of item features and user preferences, and a heuristic of adding artificial users who like the group of items, without modifying a recommendation algorithm.

I nie próbuj nic zrozumieć,
Nie pochodzi mieć od umieć.

Postmodernizm, J. Kaczmarski [2]

2 Prediction of unknown data

In this chapter I will briefly overview key concepts and methodologies in the taken approach to prediction. The presented part of the prediction domain is a foundation to the methods described in the rest of the work.

In statistical data analysis two large subfields are distinguished: exploration and prediction. To exploration we count methods that help to understand the structure of data, such as: summary statistics, various kinds of visualizations, grouping data (clustering), looking for interesting projections. Overviews of exploratory techniques can be found in [Tuk77, Pha06, Bri39]. Exploration is related to the domain of human-computer interaction and there are elements of subjectivity in it.

Prediction, in turn, is deducing about an unknown part of the world, based on a known part of the world. We can try to define prediction more precisely, encompassing most cases: we have a set of data, which we know in its entirety, and a second set of data, which we know only partially. We need to predict the unknown portion of the second set, as accurately as possible. By that it is necessary to choose an evaluation criterion, that is, to specify how to compare accuracy of different methods. Depending on the gathered data and on the situation where a prediction method is applied, determining how to properly evaluate prediction methods can be a challenge. Often needed is domain adaptation, that is adapting a method learned on a sample from one population to make predictions for another population, from which we have a smaller sample or, more commonly, from which we do not have any data gathered (a common situation in observational studies). Sometimes it is useful to consider underlying causal effects, when some of the observed variables have a causal influence on other variables and on the output variable, which value we may want to optimize or influence (we never exactly know the correct generative model for the training and test data, which encompasses all usable for prediction information about causal effects). Usage of prediction can in some cases resemble closed-loop control systems, when the predictions made influence back the scrutinized system and affect which data will be generated and used for prediction in the future. For example, recommender systems calculate lists of recommendations, which are then shown to a user, and the gathered user's preferences for the newly shown items are used to calculate future recommendations. Especially in recommender systems based on implicit user feedback important is understanding and modelling the mechanism of missing data that stems from the recommender system navigation.

It is not always possible to specify the right evaluation criterion precisely, and to properly compare methods based only on the possessed data, but staying with predicting the observed data is convenient, because it gives us opportunity to compare methods relatively precisely with the chosen evaluation criterion on held-out data. If the used dataset is large enough, we can speak about objective, impartial comparison of prediction methods (up to a certain number of compared methods). If we stay with the methodology of predicting left-out observations, we need to be aware that the obtained prediction methods need to be eventually adapted to be useful for the real prediction goal. In fact, other situations, when no test set is available (on which methods can be reliably tested

[2] "And don't try to understand anything, to own doesn't come from to know."

by comparing hold-out error) are more vague, and rarely there is a point to look for best possible accuracy (as in the methods presented in this work), when the right, maximally useful prediction task is not exactly known.

When attempting to use data for prediction one needs to have in mind, that the gathered data may come from a different distribution than the data needed to perform and evaluate predictions. In such cases we are inferring about the whole function based on observing its values only in part of its domain, for example, inferring about what is on a whole photo, on the basis on observing only a small part of it. For some datasets and tasks useful predictions can be obtained this way. For other datasets, using the gathered data to obtain useful predictions may be difficult. To decide, what is the right way to use the data, and if the acquired dataset is good enough (or if we need to gather different kind of data), one has to use domain knowledge and common sense.

Two larger subtasks in prediction are: model identification and estimation of parameters of the model on basis of observed data. Prediction is generally a more strict area than exploration. Especially the part of inferring parameters with a known probabilistic model is relatively well understood (see the standard references on machine learning grounded in Bayesian statistics [Bis06, Mac03]).

This work focuses on a prediction task, but there will be also elements of exploration. For example, the overall methodology of this study is to draw conclusions from various summaries and visualizations to iteratively improve models with the discovered effects, patterns, corrected assumptions. Thus the overall approach is closer to the way of exploratory data analysis than to the strictly Bayesian way of prediction, where the entire prior distribution is specified once, before looking at the data, and the data is not reused.

The chosen task motivates focus on the static aspect of prediction, that is, using observed data to infer about unobserved data, without variability of the model over time. For modelling movie ratings from a recommender system, the dynamic effects, that is, changing in time, turn out to have much less importance than static ones. To give an idea of the taken approach to prediction that has proved to be effective in the Netflix task, let's list the most useful methods and concepts, which also will be described in more detail in the next chapters. If you look at the description of experiments and results, it was empirically found that the best accuracy was obtained on the basis of regression methods, dimensionality reduction, an appropriate choice of regularization, integrating different models with each other, approximating the Bayesian approach in different ways, and also using several methods distant from probabilistic modelling, such as K-nearest neighbors. Important was taking a time-effective engineering approach to model identification, maintaining an ensemble of predictions, looking for and using various effects or interactions in the data, in parts of models, or in the side-results of methods.

As for the scope of dynamic modelling, time series methods were used only to a limited extent. Mainly, one-day user effects were modelled. Besides that, also were used: corrections of distance-based methods for time information, occasional use of exponential smoothing, and the simple method of binning, mostly for global biases. Some subdomains of time-dependent modelling, that are commonly used in other applications, like stochastic differential equations, causal modelling, stochastic control, were not used here for the Netflix task.

In the chapter I first describe selected general issues in prediction, by discussing in section 2.1 the decision-theoretic understanding of the task of prediction. In a static situation the decision-theoretic approach is reduced to choosing a loss function and a family of functions approximating the predicted variable, and then maximizing the posterior expected loss. There is depth in the simply put task of inference from data to obtain best prediction. Attempts to solve these kinds of tasks led to developing a wide variety of prediction methods.

Section 2.2 discusses the most popular method of prediction, linear regression. On the example of linear regression we present various concepts that recur also in any other models: model identification, compliance of the data with assumptions of the model, transformations of variables, statistical significance, feature selection, outliers removal. Non-linear variations of regression, such as logistic or Poisson regression, can be realized by iterating linear regression with weighted observations.

Section 2.3 discusses the important issue of regularization. Linear regression can be understood as maximum likelihood estimation (maximum a-posteriori estimation for flat a-priori distribution) of a simple linear model. Experience shows that, instead of maximum likelihood estimation usually it is much better to use Bayesian methods with a prior distribution that is not flat, but concentrated around a certain value. We will describe regularized variations of linear regression: ridge regression, and a more complex variant of Bayesian linear regression with prior distribution of parameters and a known error covariance matrix. We will also consider the case of uncertainty in predictors.

In section 2.4 I examine the model selection problem when choosing between two specified models, or when little is known about the real model that generated the data. I will justify, but also look critically at the holdout validation method, which is extensively used thorough the work.

In section 2.5 I discuss when small accuracy improvements are important. A commonly encountered setting is when similar prediction decisions are made multiple times. With thousands or millions of repetitions importance of prediction accuracy is greatly amplified.

2.1 Decision theory

Decision theory describes situations of optimal decision making under uncertainty. Among various proposals of capturing uncertainty the most basic, fundamental, correct one is the theory of probability, hence the approach to decision theory described here is probability based. It comes down to minimizing the expected value of the chosen loss function. Equivalently, you can maximize the expected utility function.

In the decision-theoretic approach to prediction, to make an optimal prediction means to choose a point estimate of the predicted variable (or variables) that minimizes the posterior expected loss. "Posterior" means, after observing the data. According to the task under scrutiny we should choose an appropriate loss function that specifies how to penalize different types of error of the approximation.

More precisely, here is the most common setting: we have two random variables, p-dimensional $\mathbf{X}$ and one-dimensional Y. We have a set of n independent samples from $(\mathbf{X}, Y)$, which we call training data $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$. Now there is a situation of prediction: we get a new observation, a realization of the variable $\mathbf{X}$, but we don't know the corresponding realization of $Y|\mathbf{X}$. To simplify, we will assume here that both the training data and the predicted test observation come from the same distribution, described by the density $p(\mathbf{x}, y|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of parameters with a prior distribution $p(\boldsymbol{\theta})$. The observed pairs $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$ let us infer the posterior distribution $p(\boldsymbol{\theta}|D)$, used to make predictions according to $p(y|\mathbf{x}, \boldsymbol{\theta})$. More complex scenarios of domain adaptation are often considered, like covariate shift situations [Shi00, Sug07, Sug08], where the training set and test set distributions differ, we can consider also possible causal mechanisms [Pea09], various kinds of missing data in the training set, or we can consider semi-supervised learning settings, where we have additional unlabeled data (only samples from $\mathbf{X}$).

We want to predict the unknown $Y|\mathbf{X}$ as accurately as possible. What does it mean, as accurately as possible? In the decision-theoretic approach we seek to approximate $Y|\mathbf{X}$ with a function from a family $f_\alpha(\mathbf{X})$, and select as the optimal the function that minimizes expected posterior loss, for a chosen loss function. We can consider parametric approximation, or nonparametric, where the function is free-form.

In the parametric approximation we choose the approximating function from a fixed family of functions $\hat{y}(\mathbf{x}, \alpha)$. We select the parametrization α that minimizes the expected posterior loss:

$$\underset{\alpha}{\operatorname{argmin}} \int l(y, \hat{y}(\mathbf{x}, \alpha)) dp(\mathbf{x}, y|D)$$

In the nonparametric case, we select the minimizing function in a free-form $\hat{y}(\mathbf{x})$:

$$\underset{\hat{y}}{\operatorname{argmin}} \int l(y, \hat{y}(\mathbf{x})) dp(\mathbf{x}, y|D) \tag{1}$$

The loss function $l(y_1, y_2)$ is chosen depending on the application. Choice of loss function is a non-obvious task itself, another source of uncertainty next to identification of the probabilistic model. Often used here is a quadratic loss function. It is worth to examine it closer, because the task of Netflix Prize, on which we focus here, is to minimize the RMSE (root mean squared error) on the test set, which is close to minimizing the MSE (mean squared error). MSE on a random held-out test set is an unbiased estimator of the expected quadratic loss, if we treat the test data as not being part of the training data. Minimizing the expected posterior quadratic loss leads to:

$$\underset{\hat{y}}{\operatorname{argmin}} \int (y - \hat{y}(\mathbf{x}))^2 dp(\mathbf{x}, y|D) = \underset{\hat{y}}{\operatorname{argmin}} \int \int (y - \hat{y}(\mathbf{x}))^2 p(\mathbf{x}|D) p(y|\mathbf{x}, D) d\mathbf{x} dy$$

$$= \underset{\hat{y}}{\operatorname{argmin}} \int F(\mathbf{x}, \hat{y}(\mathbf{x})) d\mathbf{x}$$

where

$$F(\mathbf{x}, \hat{y}(\mathbf{x})) = \int (y - \hat{y}(\mathbf{x}))^2 p(\mathbf{x}|D) p(y|\mathbf{x}, D) dy$$

To find the optimal $\hat{y}(x)$, we set the derivative of F to zero.

$$\frac{\partial F(\mathbf{x}, \hat{y}(\mathbf{x}))}{\partial \hat{y}(\mathbf{x})} = 2p(\mathbf{x}|D)\Big(\hat{y}(\mathbf{x}) - \int y p(y|\mathbf{x}, D) dy\Big) = 0$$

$$\hat{y}(\mathbf{x}) = \int y p(y|\mathbf{x}, D) dy = E(y|\mathbf{x}, D) \tag{2}$$

So the resulting optimal point prediction is the expected value of the output variable.

To have a broader insight into different possible situations in prediction, let's consider a different loss function than quadratic, namely the example of binary classification – when Y takes one of two possible values. We will look at a largely simplified example of medical diagnostic tests (ignoring the domain adaptation issues). Each test is described by two probabilities: the probability of a correct guess when a person is ill, and the probability of a correct guess when a person is healthy. Our decision problem is to select the best test among multiple available. To select the test that will accurately detect a few sick people in a large population of healthy people, we should choose an asymmetric loss function: the value of the loss function for type I error, that is misclassifying a sick patient as healthy, should be greater than type II error, that is misclassifying a healthy patient as sick.

We can conclude, that in prediction it is not enough to have a good probabilistic model for data. We need to choose the right loss function, and to do it we need to know the context of the data analysis. Often necessary is expert's knowledge in the area where the data comes from.

After establishing the loss function, to make predictions by formula (1) we need the distribution $p(\mathbf{x}, y|D) = \int p(\mathbf{x}, y|\theta) dp(\theta|D)$. In practice we do not know the exact process that generated the data, that is, we do not know the model $p(\mathbf{x}, y, \theta)$ and usually the

biggest challenge in prediction is to identify the model accurately enough (note that some aspects of the model can be irrelevant to the chosen loss function, and hence, do not need to be identified accurately). Similarly difficult are also attempts to model the expected posterior loss (1) directly, bypassing the explicit construction of the full generative model.

Even if we identify the model fairly accurately, calculating posterior probabilities often has high computational complexity, which forces us to use approximations, introducing further inaccuracies.

Let's say a bit more about the two aforementioned approaches for prediction. Prediction methods can be divided into two basic types: an indirect approach, called generative, and a direct approach, in the case of classification tasks (when Y is a categorical variable) called discriminative. The generative approach is to calculate $p(\mathbf{x}, y|D)$ (the posterior probabilities after taking into account the training data), using Bayes' rule, and then to choose the approximation of $y|\mathbf{x}$ that maximizes (1).

Because our loss function is focused on approximating the output variable y, often a fully generative model $p(\mathbf{x}, y|D)$ of all observed variables is not really needed. It often may be enough to discover the general relationship $p(y|\mathbf{x}, D)$ without exact modelling the distribution of the predictor variables $p(\mathbf{x}|D)$ (we can also model the expected posterior loss directly, as suggested by the formula (2), without explicitly computing the posterior probabilities as an intermediate step). This direct approach (in classification tasks called discriminative), because it does not need to model the part of data unrelated to the chosen prediction task, results in methods with fewer parameters to be learned than the corresponding generative methods. An example can be binary classification of a variable drawn from one of two multidimensional Gaussian distributions. The generative approach leads to methods like LDA or QDA with $O(k^2)$ parameters, where k is the dimension of the Gaussians (the number of predictor variables). The direct approach leads to logistic regression methods with only $O(k)$ parameters.

If we knew the probabilistic model exactly, the generative approach is equivalent to the direct (discriminative) approach. In practice, we usually have limited capability of identifying the part $p(\mathbf{x}|D)$, and trying to model that part takes effort, and can introduce inaccuracies. Sometimes it is useful to consider generative models, but I almost always tend to use the direct approach.

A whole variety of methods realize the direct approach and the generative approach. For different structures of hidden variables and parameters (which can be seen as hidden variables), as well as for various types of uncertainty about the form of the model that generated the data, one can propose different approximations. If we need very accurate prediction, we can use MCMC or Variational Bayesian approximation. Many methods can be regarded as high-level approximations of the Bayesian approach, for example, methods inspired by biology, such as neural networks, or methods inspired by statistical physics, such as Boltzmann machines. There are also effective methods, for which the connection to probability is not obvious, such as K-nearest neighbors.

The next section describes the most popular and best-examined prediction method, linear regression.

2.2 Linear regression

Prediction methods can differ considerably depending on the type of variable predicted. When Y has only several possible values, the prediction task is called classification, and when Y is continuous, the task is called regression. The distinction between regression and classification is not always clear, for example, the logistic regression method, which models the probability of Y taking one of two values, is sometimes called binary classification. The type of predicted variable influences the decision problem to optimize and the choice of learning algorithm, for example, it can influence the structure of unobserved variables,

prior distributions, the inference method, the computational complexity, or whether the optimization problem has local minima to avoid.

I will discuss briefly linear regression, which is the most widely used method of prediction, while being one of the simplest and most thoroughly studied. Linear regression owes its popularity to the prevalence of the normal distribution in nature, which is a consequence of the fact, that sums of independent variables, when the number of variables increases, converge (apart from a few exceptions) to a normal distribution.

The first known use of multivariate linear regression was by Gauss, who used this method to estimate the path of the dwarf planet Ceres using small number of observations [Gauss1809, Ste95]. To solve the resulting system of linear equations he used the algorithm known today under the name of Gauss elimination.

In linear regression we assume a linear relationship between a vector of p constants $\mathbf{x}_i$, called explanatory variables, covariates or predictors, and a variable y_i, called the response variable, predicted variable, or output variable.

The linear regression model has the form:

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where $\mathbf{y}$ is a vector of n predicted outputs, X is a matrix of $n \times p$ constant predictors, $\boldsymbol{\beta}$ is a vector of p unknown coefficients (parameters), and $\boldsymbol{\varepsilon}$ is a vectors of errors, about which we assume that they are independent, and come from a normal distribution with equal variance. Put in another way, $\mathbf{y} \sim N(X\boldsymbol{\beta}, \sigma^2\mathbf{I}_n)$, where $\mathbf{I}_n$ is an identity $n \times n$ matrix.

The basic task in the linear regression model is to estimate the unknown coefficients $\boldsymbol{\beta}$, which are then used for prediction when we have observed $\mathbf{x}_{new}$, but we do not know, and want to estimate the y_{new}.

$\boldsymbol{\beta}$ can be estimated by least squares method, minimizing the sum of squared errors $(\mathbf{y} - X\boldsymbol{\beta})^T(\mathbf{y} - X\boldsymbol{\beta})$. Another view, giving the same point estimators for $\boldsymbol{\beta}$, is the maximum likelihood method, that is assuming a flat prior distribution $p(\boldsymbol{\beta}) \propto 1$ in the Bayesian approach to parameter inference, and taking the maximum a-posteriori point estimate of $\boldsymbol{\beta}$, where the posterior distribution is calculated using Bayes' rule: $p(\boldsymbol{\beta}|X, \mathbf{y}) \propto p(\mathbf{y}|X, \boldsymbol{\beta})p(\boldsymbol{\beta}) = p(\mathbf{y}|X, \boldsymbol{\beta})$. Then $\boldsymbol{\beta} \sim N(E\boldsymbol{\beta}, Var\ \boldsymbol{\beta})$, where:

$$E\boldsymbol{\beta} = (X^TX)^{-1}X^T\mathbf{y}$$

$$Var\ \boldsymbol{\beta} = (X^TX)^{-1}\sigma^2$$

When applying regression to real life data coming from an unknown distribution necessary is regression diagnostics [Far04, Far06], that is making sure that the model assumptions are sufficiently met by the data. Residuals should be examined to test if the errors are normally distributed, independent, and homoscedastic (have constant variance). Standard plots used for the above purpose are: residuals vs. fitted values, quantile-quantile plots of residuals vs. normal distribution, plotting square root of standardized residuals vs. fitted values, or autocorrelation plots for time dependence. We should check whether the relationship between the predictors and the response is really linear, and whether eventual transformations of the response and predictors should be performed (Box-Cox procedure can suggest transformations of the response, and generalized additive models can suggest transformations of the predictors). Sometimes it is useful to standardize (normalize) the predictor variables, that means, to subtract the estimated mean, and divide by the estimated standard deviation. Another thing to do is removing or otherwise dealing with outliers, that is unusual observations, which visibly do not come from the assumed model (often outliers result from errors in the process of creating the dataset). Special care should be taken to examine influential observations – which removal considerably influences the regression coefficients. Plots usually used to look for outliers are: residuals vs. leverage, Cook's distance vs. leverage, and one-dimensional boxplots.

In an ordinary, non-regularized linear regression usually feature selection improves generalization performance. Commonly used criteria for selecting or rejecting predictors are:

- statistical significance (paying attention to multicollinearity)
- comparing Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) of models. BIC is $p \log n - 2 \log L$, where L is the data likelihood, p is the number of parameters, and n is the number of observations. AIC is $2p - 2 \log L$.
- change in residual sum of squares (RSS)

A criterion that is sometimes used, but should be avoided (often gives misleading information about the dependence between two variables) is the simple Pearson correlation coefficient between the response and a predictor.

To roughly assess goodness of fit, proportion of the explained variance to the total variance can be used: $R^2 = (TSS - RSS)/TSS = 1 - \sum_i (\hat{y}_i - y_i)^2 / \sum_i (y_i - \bar{y})^2 = \sum_i (\hat{y}_i - \bar{y})^2 / \sum_i (y_i - \bar{y})^2$, where $\hat{y}_i$ are the predictions, y_i are the observed values, and $\bar{y}$ is the average value of y_i.

In a situation of non-constant variance of errors (heteroscedasticity) weighted linear regression can be used:

$$\bar{\boldsymbol{\beta}} = (X^T W X)^{-1} X^T W y$$

where W is a diagonal matrix of weights $w_{ii} = 1/\sigma_i^2$. Regression with a non-Gaussian response variable is called generalized linear regression, and is realized by iteratively reweighted least squares – running several iterations of weighted linear regression with weights w_{ii} dependent on the current estimates $\hat{y}_i$. For a binary response it is logistic regression with $w_{ii} = \hat{y}(1 - \hat{y})$. For a Poisson response it is Poisson regression with $w_{ii} = \exp(\hat{y})$. We could also use an additional regression component to model the varying variance of errors, with the same predictors that are used to model the mean, or with a different set of predictors.

Ordinary linear regression and generalized linear regression are results of the maximum likelihood method. Experience shows that maximum likelihood estimation often results in overfitting the training data, meaning that the method fits the training data closely, but does not generalize well on the unobserved data (as observed by increased error on held-out test data). Instead of using a flat prior distribution on parameters $\boldsymbol{\beta}$, it is usually better to use a prior centered around some value, leading to regularized estimates.

2.3 Regularization

Using a wrong model or wrong prior distributions for parameters usually leads to suboptimal accuracy (it may be visible as overfitting or underfitting the data). The last section introduced the method of linear regression, which is a result of maximum likelihood estimation, that is MAP (maximum a-posteriori) estimation with assumed flat distribution on parameters. We can list many drawbacks of flat priors: they do not appear in nature (they are improper priors – are not probability distributions), and using them usually leads to overfitting the data and to inferior accuracy in comparison to priors concentrated around some value. The most common distribution in nature is normal distribution and this prior distribution seems the to be most natural choice for parameters β in regression (we should note that a normal distribution with a large variance will have a similar effect to using a flat prior).

Empirically, we usually observe that, in comparison with the maximum likelihood method, a technique called regularization improves predictive accuracy. Regularization, called sometimes weight decay or parameter shrinkage, adds to the likelihood function a term penalizing large weights. The most common form of regularized linear regression,

called ridge regression, uses L^2 penalty for weights (also known as Tikhonov regularization), resulting in the following formula:

$$\boldsymbol{\beta} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}$$

The L^2 regularization comes from MAP estimation in a linear regression model with Gaussian priors on parameters β, and with a known, fixed Gaussian error term. Other types of regularization are less common, as, for example, L^1 regularization, which gives a method called lasso regression.

We can select the optimal λ, for example, by cross-validation, that is by randomly choosing a subset of observations as a held-out test set, and learning the parameters β on the training set containing the remaining observations. The procedure is repeated multiple times and the resulting accuracy is averaged, which gives an estimate of the predictive accuracy on unknown data (see also the next section for discussion of cross-validation). When λ is lower than the optimum, we typically observe a phenomenon of overfitting: with a reduction of λ the training error decreases, but the error on the held-out test set increases. When λ is larger than the optimal value, we observe underfitting: the estimated parameters will be closer to zero, and both the training error and the test error increase.

Ridge regression bases on an a-priori belief that the values close to zero are more likely than the values distant from zero. Ridge regression is the result of Bayesian inference in the following probabilistic model, where $\mathbf{y}$ depends on X linearly through the unknown parameters β plus white noise with fixed, point estimated variance σ^2:

$$\mathbf{y} \sim N(X\boldsymbol{\beta}, \sigma^2 I) \tag{3}$$

where a Gaussian prior distribution for the parameters $\boldsymbol{\beta}$ is assumed:

$$\boldsymbol{\beta}^{a-priori} \sim N(0, \tau^2 I) \tag{4}$$

Using Bayes' rule $p(\boldsymbol{\beta}|X, \mathbf{y}) \propto p(\mathbf{y}|X, \boldsymbol{\beta})p(\boldsymbol{\beta})$ we calculate the posterior distribution of $\boldsymbol{\beta}$ after taking into account the observed data X, $\mathbf{y}$:

$$\boldsymbol{\beta} \sim N(E\boldsymbol{\beta}, Var\ \boldsymbol{\beta})$$

$$E\boldsymbol{\beta} = (X^T X + \frac{\sigma^2}{\tau^2} I)^{-1} X^T y$$

$$Var\ \boldsymbol{\beta} = (\frac{1}{\sigma^2} X^T X + \frac{1}{\tau^2} I)^{-1}$$

We see that the obtained PME (posterior mean) or MAP estimate of $\boldsymbol{\beta}$ is equivalent to ridge regression with $\lambda = \sigma^2/\tau^2$.

If there are more features (columns of X) than observations (rows of X), faster to calculate is the dual form of the ridge regression, kernel ridge regression – PME of the corresponding Gaussian process:

$$Ey_{new} = \boldsymbol{x}_{new}^T X^T (XX^T + \lambda I)^{-1} \boldsymbol{y}$$

The above simple probabilistic linear model (ridge regression) performs well enough for a wide choice of data and prediction tasks, even when the underlying assumptions are not exactly met, but often a more flexible model is needed. Common modifications are:

- other prior distribution of parameters, with mean other than the vector zero, and with parameters not independent a-priori, but with the dependence structure defined by a covariance matrix S

$$\boldsymbol{\beta}^{a-priori} \sim N(\boldsymbol{\beta}_0, S) \tag{5}$$

- heteroscedastic errors (varied variance - can be seen as weighting of observations) and not independent, with the dependence structure defined by a covariance matrix Ω

$$\boldsymbol{y} \sim N(X\boldsymbol{\beta}, \Omega) \tag{6}$$

In the simplest setting S, Ω are constant matrices, and X also remains constant. As $W = \Omega^{-1}$ we denote the precision matrix of errors. Then:

$$\boldsymbol{\beta} \sim N(E\boldsymbol{\beta}, Var\ \boldsymbol{\beta})$$

$$E\boldsymbol{\beta} = (X^T W X + S^{-1})^{-1}(X^T W y + S^{-1}\boldsymbol{\beta}_0)$$

$$Var\ \boldsymbol{\beta} = (X^T W X + S^{-1})^{-1}$$

The above model is very close to the Black-Litterman model used in portfolio optimization [Bla92, Pal08]. Again, predictions in the dual version are:

$$Ey_{new} = \boldsymbol{x}_{new}^T[\boldsymbol{\beta}_0 + SX^T(XSX^T + \Omega)^{-1}(\boldsymbol{y} - X\boldsymbol{\beta}_0)]$$

The parameters S, Ω in (5), (6) were constant matrices, and τ^2 and σ^2 were constants in the model (3), (4). Instead of using fixed constants or point estimators, these parameters can be treated as random variables with defined priors (for S and τ^2 called hyperpriors). Those additional priors could be estimated, for example, from similar tasks in a multitask learning setting.

All of the above is not enough to obtain best accuracy in matrix factorizations, which can be seen as performing simultaneous ridge regressions: one regression for each user, and one regression for each movie. The matrix of predictors X is not constant in those regressions, but consists of random variables with different amount of uncertainty. It was noticed experimentally [Fun06], that better than a constant regularization term is using an amount of regularization growing linearly with the number of observations. Variational Bayesian matrix factorization [Lim07, Rai07] explains to some extent this phenomenon – the influence of uncertainties from each observation sum up to a term that behaves as additional regularization. A well performing modification of ridge regression for the case of non-constant X entries contained both the linear and the constant regularization term:

$$\boldsymbol{\beta} = (X^T X + (\lambda_1 + \lambda_2 n)I)^{-1} X^T \mathbf{y}$$

where n is the number of observations (rows of X).

2.4 Model selection

In the previous sections we have presented the basic theory of prediction: calculating the posterior distribution of the parameters after observing the data, using Bayes' rule $p(\theta|D) \propto p(D|\theta)p(\theta)$, and then minimize expected loss. However, in this Bayesian approach we assume that we know the precise form of the probabilistic model that generated the data. That probabilistic model is, in fact, never exactly known for real-life data. The Bayesian approach encourages to encompass the entire uncertainty in the form of a prior distribution (before the data has been observed), but a fully Bayesian approach of course is not possible to realize in practice, for multiple reasons. Our prior distribution would have to include all distributions possible to observe for all possible data types, categorical and continuous. Not to mention that such a huge distribution would have to contain domain knowledge from all areas that may generate the data. Even in narrower, specialized cases, for example, when we know that a good approximation of the data is a multidimensional linear model, and we know the area from which the data originates, it is clear that many matters remain that are difficult to be expressed with a prior distribution. Let's list some

of the common decisions to make when using a linear model: the choice of input variables (predictors) and interactions between input variables, transformations of input and output variables, weighting observations, the choice of prior distribution for parameters. A large challenge is that domain knowledge is often intuitively expressed (or even unconscious) and difficult to translate into a probabilistic model.

A practical approach to prediction for real-life data is using the data repeatedly, iteratively improving the model. Limited reuse of data should not cause noticeable overfitting of the data (choosing a model with worse performance on unseen data). Having more data allows to choose between more models, that is why larger datasets are interesting for researchers, allowing to use a wider selection of techniques than when working with small datasets, and allowing to better understand the underlying process that generated the data.

The process of searching for the best model can be treated in simplification as a series of model selection tasks, that is, we are faced multiple times with a choice between two or more models (sometimes infinitely many). The choice of models for comparison may have different origins: it may result from domain knowledge, from hypothesizing about possible effects in data, or from summarization or visual examination of data, noticed patterns, dependencies, probability distributions, incompliances with the model assumed so far, etc.

In the model selection process, usually smaller and simpler models are preferred (a heuristic rule called Occam's razor, sometimes formalized as a minimum description length principle). An a-priori assumption that simpler models are more probable than complex ones would be disputable. Nature sometimes favors simplicity, for example, isolated physical systems approach thermodynamic equilibrium, and sometimes nature favors complexity, for example, simple rules can lead to complicated fractal structures [Man82, Wol02]. Independently of whether we a-priori expect a simple or a complex model, if we consider concrete situations of model selection between similar models with a certain amount of observed data coming from one of the models, then either calculating evidence of each model (and the Bayes Factor), or considering a chosen measure of prediction error lead to model selection criteria like AIC or BIC, which penalize larger models. Besides better predictive accuracy, there are many engineering reasons to prefer simpler models – they are easier to work with, more likely have a tractable algorithm for parameter inference, need less code, are less time-consuming to implement, and are less prone to implementation errors. On the other hand, we can mention some advantages of larger models – they are useful for model identification, can help to spot important effects which smaller models are not capable to capture. We should remember that a choice of one of many models is a simplification, valid when the remaining models are very implausible. Sometimes proper Bayesian averaging of models is needed, instead of selecting one model.

I will examine a situation of model selection between two fixed, simple models M_1, M_2. Assume we have N samples of data D, generated iid (independent, identically distributed) from one of two models (distributions), with the a-priori probability of the models: $p(M_1)$, and $p(M_2) = 1 - p(M_1)$. The posterior distribution is thus following:

$$p(M_1|D) = \frac{p(D|M_1)p(M_1)}{p(D|M_1)p(M_1) + p(D|M_2)p(M_2)} = \frac{1}{1 + \frac{p(D|M_2)p(M_2)}{p(D|M_1)p(M_1)}}$$

The value $BF = \frac{p(D|M_2)p(M_2)}{p(D|M_1)p(M_1)}$ is called Bayes Factor [Mac03, Bis06]. Some recommend model selection according to a fixed threshold $BF > \alpha$, but there is also criticism of that approach, because the decision by BF is independent of the context, does not take into account the actual purpose of model selection [Gel11, Gel95].

Because the choice between M_1 and M_2 is a situation of decision making, it is justified to treat such a meta-task in accordance with the decision theory, where we choose a loss function $l(M_{choice}, M_{real})$ for each of the two possible guessed models $M_{choice} = d(D)$,

and each of the two possible actual drawn models M_{real}. The optimal decision is given by minimizing the expected posterior loss:

$$argmin_d \; l(d(D), M_1) * p(M_1|D) + l(d(D), M_2) * p(M_2|D) \tag{7}$$

The optimal decision function $d(D)$ depends on the choice of the loss function l, which states the cost of the errors of the first and second type, that is the cost of selecting the wrong model in our meta-task. Let's denote the values of loss function for the four possible arguments by: $l_{11}, l_{12}, l_{21}, l_{22}$, where the first subscript indicates the model selected, and the second denotes the actual model, which generated the data D. A reasonable assumption about our loss function is that the correct recognition of the model is better than the incorrect, that is $l_{11} < l_{21}$ and $l_{22} < l_{12}$. Minimizing the expected posterior loss (7) gives a decision rule selecting the model M_1 iff $\frac{p(M_2|D)}{p(M_1|D)} < \frac{l_{21}-l_{11}}{l_{12}-l_{22}}$. We see that the decision-theoretic approach to the meta-task of a choice between two known models is equivalent to using Bayes Factor for a cut-off threshold dependent on the chosen loss function.

The loss function in the meta-task can be directly related to the loss function used to evaluate the generalization error. For example, in the Netflix Prize task we are interested in expected RMSE on unseen data (from the distribution of the observed data), so if we want to compare two models and choose one of them, it seems reasonable to use the expected RMSE also as the loss function in the meta-task of model selection. Because for real-life data we do not know the probabilistic model that generated the data, especially we do not know much about it when we begin the analysis of a given dataset, useful are universal criteria, same for all models. In practice, popular universal model selection criteria are methods comparing the validation error (here the average loss function) on held-out validation data (a test set). Widely used is k-fold cross-validation, where the data is divided into k equal parts, each of them serves one time as the test set (the remaining parts form the training set), and the final result is the average validation error from the k-times learned model. An advantage of methods based on hold-out validation error is their simplicity and versatility. Such methods are the same for different datasets, even for a completely unknown model.

Having different possible criteria of model selection, one can wonder, when one criterion is better than another, by how much better, and what "better" means. We will compare, for a choice between two fixed simple models, two groups of model selection criteria: one group is using Bayes Factor for different thresholds (equivalent to choosing a loss function in the meta-task of deciding on one model), and the second group are universal, model-independent criteria measuring validation error. For those model selection criteria we will compare the frequency of type I and type II errors, that is, how often the given criterion identifies the correct model. In this toy study we evaluate model selection criteria according to how well they select a more probable model, but instead we could, for example, evaluate model selection criteria according to how well they select a model that has better predictive accuracy (for a given measure of predictive accuracy).

Let's look closer at a case of model selection between two known simple models. The accuracy of different model selection criteria will be judged by frequency of making type I and type II errors, that is, how often a given criterion is wrong and chooses M_2, when the data comes from M_1, and how often it chooses model M_1, when the data comes from M_2. The data is generated as follows: first with probability $\frac{1}{2}$ we choose the model M_1 or M_2 (we assume, that the models M_1 and M_2 are a-priori equally likely), then we generate the parameter $\mu \sim N(0, 1)$ for the model M_1 or two independent parameters $\mu_1 \sim N(0, 1), \mu_2 \sim N(0, 1)$ for the model M_2, and generate N data samples from the selected model. In the model M_1, N iid samples are drawn from the distribution $N(\mu, 1)$. In the model M_2, $\frac{N}{2}$ iid samples are drawn from the distribution $N(\mu_1, 1)$, and $\frac{N}{2}$ iid samples are drawn from the distribution $N(\mu_2, 1)$.

The evidence function for the model M_1 is following:

$$p(D|M_1) = \int p(D|\mu, M_1)p(\mu, M_1)d\mu = (2\pi)^{\frac{N}{2}}\sqrt{N+1}\exp\Big(\frac{1}{2}\frac{(\sum_{i=1}^{N} y_i)^2}{N+1} - \frac{1}{2}\sum_{i=1}^{N} y_i^2\Big)$$

The evidence function for the model M_2:

$$p(D|M_2) = (2\pi)^{\frac{N}{2}}(\frac{N}{2}+1)\exp\Big(\frac{1}{2}\frac{((\sum_{i=1}^{\frac{N}{2}} y_i)^2 + (\sum_{i=\frac{N}{2}+1}^{N} y_i)^2)}{\frac{N}{2}+1} - \frac{1}{2}\sum_{i=1}^{N} y_i^2\Big)$$

We assumed, that the models are a-priori equally probable: $p(M_1) = p(M_2) = \frac{1}{2}$. Then Bayes Factor is the ratio of M_2 evidence and M_1 evidence:

$$\frac{p(D|M_2)}{p(D|M_1)} = \frac{\frac{N}{2}+1}{\sqrt{N+1}}\exp\Big(\frac{1}{2}\frac{(\sum_{i=1}^{\frac{N}{2}} y_i)^2 + (\sum_{i=\frac{N}{2}+1}^{N} y_i)^2}{\frac{N}{2}+1} - \frac{1}{2}\frac{(\sum_{i=1}^{N} y_i)^2)}{N+1}\Big)$$

Figures 1 and 2 show the precision-recall relation for predicting the model M_1 using different model selection criteria. Bayes Factor with changing cut-off threshold level (BF on the plots denotes $\log\frac{p(D|M_1)}{p(D|M_2)}$) is compared with several holdout validation methods:

- VAL20% - random 20% of data as a validation set, with the remaining 80% used as the training data,
- VAL50% - random 50% of data as a validation set,
- VAL20%×5 - averaged result of 5 draws of VAL20%,
- VAL50%×5 - averaged result of 5 draws of VAL50%,
- CV5FOLD - 5-fold cross-validation,
- CV10FOLD - 10-fold cross-validation.

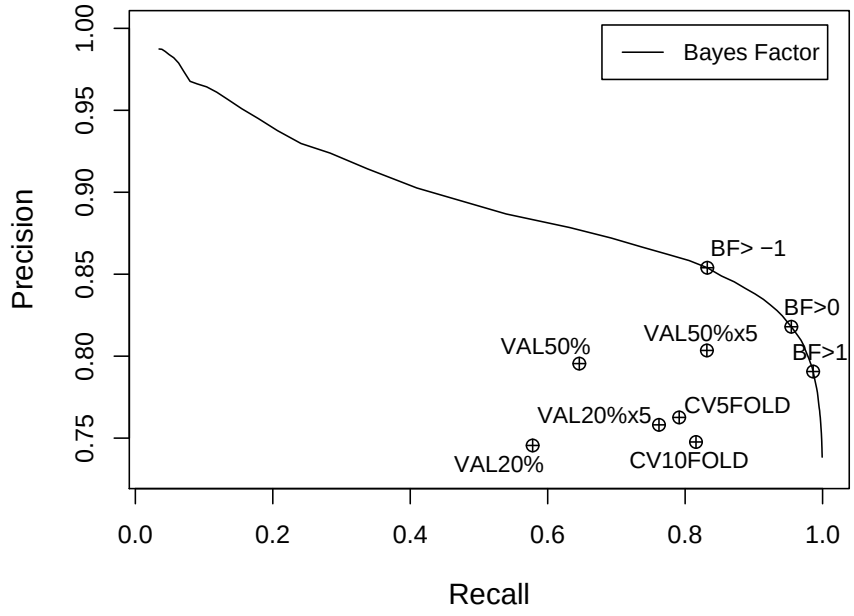
Figure 1 compares precision-recall for $N = 100$ data samples, averaged over $50,000$ draws of models and data. Figure 2 compares precision-recall for $N = 2000$ data samples, averaged over $100,000$ draws of models and data to evaluate Bayes Factor, and $10,000$ draws to evaluate the holdout validation methods.

We observe visibly worse accuracy of the methods based on the hold-out validation error, in comparison to the optimal decisions (for a desired recall or precision level) by Bayes Factor. We should remark, that for real-life data we rarely know the exact form of the models we want to compare. Methods like cross-validation are more convenient because they are identical for any data, even when the underlying model is completely unknown.

In this work the following methods were used for model selection:

- for the purpose of comparing methods RMSE_{15} was used, that is using 15% of the probe set (about 0.2% of all available ratings) as a held-out validation set; this validation set was used also for automatic and manual tuning of parameters, which can be seen as model selection among a large number of similar models,
- for the purpose of feature selection were used: calculating statistical significance in linear regression (or ridge regression) and comparing change in the sum of squared error (SSE) (see chapter 5 "Experimental results"),
- occasionally in linear regression were used criteria like AIC and BIC [Far04] (also their greedy, automated versions, stepAIC and dropterm functions in R), that is SSE (or negative double log-likelihood in general) corrected by subtracting the term Cp, where p is the number of parameters of the model, and C is a chosen constant ($C = 2$ in AIC, and $C = \log n$ in BIC).

Figure 1: Precision-recall: Cross-validation vs. Bayes Factor, N=100



A validation set of size 0.2% of the training set may seem small, even if it is about 210,000 data points for the training set with 100 million data points, but in my judgement, model assessment using this single validation set was good enough for the goals of the experiments. The validation set of this size allowed for extensive model selection, feature selection, and parameter tuning, without introducing large overfitting.

There is a difference between model selection to find the most plausible model, and model selection optimizing the selected criterion of expected loss. RMSE on a validation set has an advantage of being a good estimator of generalization RMSE (our expected loss). It is distant from being the most effective model selection criterion, but it worked well enough for us in the Netflix task.

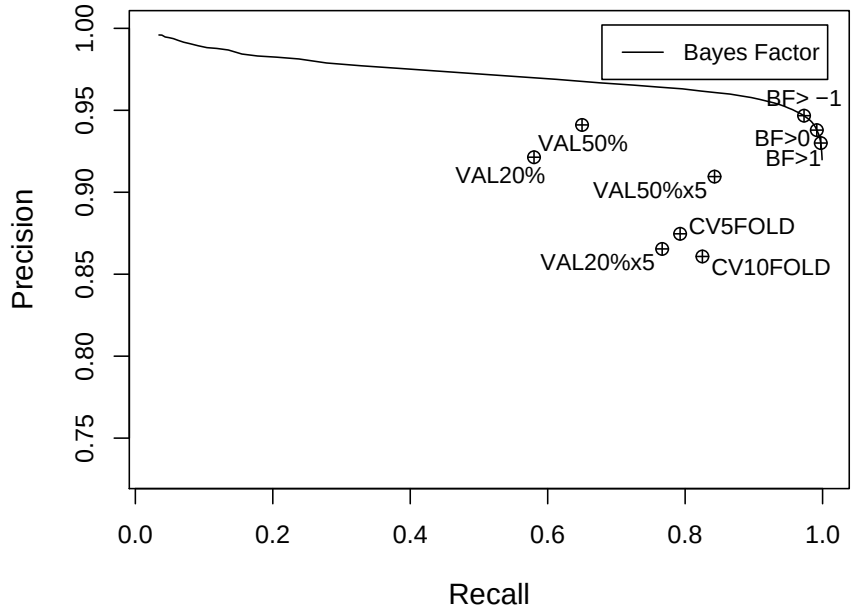
Because the amount of available data is always limited, smaller models are preferred, and it reinforces the need of model selection. We can see limitations of methods, such as based on decision trees, that perform a large number of model selection decisions, usually based on a small amount of data. Large-scale, automatic feature selection with millions of candidate features does not work well, even with very large datasets, unless we can use domain knowledge to limit the set of features considered.

2.5 Importance of accuracy

After deciding on a prediction task, and having data, on which we can evaluate predictions, we can wonder how accurate prediction we need. Depending on the application, we can distinguish different levels of needs:

1. spreadsheet-level statistics (this is the usual level of needs) – summation, multiplication, arithmetic averages,
2. one-dimensional dependence, fitting curves in 2D, one-dimensional linear regression, (or logistic, multinomial, Poisson regression, etc.),
3. multidimensional regression, maximum likelihood methods, statistical significance,
4. accurate model identification, discovering all important features, accounting for missing data effects,

Figure 2: Precision-recall: Cross-validation vs. Bayes Factor, N=2000



5. ridge regression, regularized methods,
6. special regularization, crudely approximated Bayesian modelling,
7. discovering the right probabilistic model, accurate approximations of Bayesian modelling.

Maximal obtainable accuracy of prediction may be needed, for example, in applications in finance, prediction markets, bookmaking, or when designing a compression standard (for images, video, audio, text), etc. This work is centered around obtaining possibly most accurate prediction of item ratings with the intent to be applied to calculate personalized or non-personalized recommendations, and we can wonder, how accurate prediction is really needed in applications of this kind.

Depending on the application, different degrees of recommendation quality can be considered good enough, and what follows, different levels of prediction accuracy may be required. We can grade commonly used rating prediction methods and recommendation methods according to how precisely the data is modelled:

1. lists of top items by arithmetic average (non-personalized),
2. lists of top items (non-personalized) with regularized estimates and a penalty for a small number of ratings,
3. K-NN recommender system, Pearson similarity,
4. SVD-based recommender system, non-regularized,
5. basic regularized methods, SVD with constant regularization, K-NN with regularized similarities,
6. proper linear regularization of SVD,
7. item scores adjusted for uncertainty of predictions,
8. adjustments for the missing data structure.

Each subsequent level on the list gives visibly more accurate recommendations over the previous level – except the second group of methods, which may give better recommendations (although non-personalized) over the next two levels, and except the last group of

methods (adjustment for missing data), which yet has to be tested, and the best way of adjustment has to be determined.

Because the task of proposing lists of recommendations is not precisely defined, and because usually we are limited to tune accuracy on ratings coming from the training set distribution (rated are only items chosen by the user), it can be disputed whether there is a point in striving to obtain maximum accuracy for an inexact task on an imperfect dataset. This work is focused on obtaining best accuracy for the chosen task of rating prediction evaluated by RMSE on the data distribution, treating the task as if it were fully relevant. Such extensive analysis coming from the Netflix Prize competition, besides helping to understand the domain of recommendations, may contribute to better understanding of other prediction tasks or data mining tasks.

Because also a wider scope interests us, not only recommendations and prediction of ratings, let's digress about a commonly appearing general case in prediction, repeated bets. Accuracy of prediction is typically the more important, the more times a given situation of prediction repeats (an extreme case I know is an evaluation function in computer chess, which needs to very accurately predict the likelihood of winning in a given position – executed $10^9 - 10^{12}$ times during one game). Many problems can be considered as series of similar bets. We will look at fractional bets, where as the stake we can pick any fraction of the current bankroll. Fixed-size bets with 0-1 decisions whether to enter the bet can be seen as a special case of fractional bets. The subject of repeated bets is loosely related to recommendations (recommendations can be seen as bets on user's satisfaction and user's time, but rather we cannot speak about any bankroll here). Repeated bets are a clear, common situation in finance (also personal) or gambling, where usually the notion of bankroll can be highlighted and situations of making bets of certain value can be precisely distinguished.

As repeated betting we understand the following situation of making a fractional bet, repeated N times. Each bet is associated with a certain event (Bernoulli trial), which can happen with probability p, and we can bet a fraction $r \in [0, 1]$ of the current bankroll on whether the event happens. If the event happens, we win αry, which is added to the bankroll, otherwise we lose ry. To simplify we assume that all events are independent (but remembering that in reality they usually are not), meaning that the series of events is a Bernoulli process. Each sequence of N decisions r_i leads to a certain distribution of the final bankroll, which has a certain value for the decision maker. One could try scoring the final distributions with a chosen utility function, individual for a decision maker – a person may be risk-averse or risk seeking. A common simplification is to use the expected logarithm of the final bankroll as the utility function for the decision maker. Then in each round of betting the optimal decision is the same $r_i = r$. Our utility score U is $E \log(\Delta b)$, the average growth rate in a single turn of betting:

$$U(r, \alpha, p) = p \log(1 + \alpha r) + (1 - p) \log(1 - r) \tag{8}$$

where r is a fraction of the bankroll bet, α is the reward, and p is the probability of win. Maximizing U with respect to r we get the result called Kelly criterion [Kel56] – the fraction of the bankroll should be chosen according to the rule:

$$\hat{r} = \max(0,\ (1 + \frac{1}{\alpha})p - \frac{1}{\alpha}) \tag{9}$$

In practice, we do not know the true probability of win, and in its place we can use a point estimator. Here comes the importance of prediction accuracy – we can wonder, what would happen, when our estimate of p differs from the optimum by a certain value ε. By how much U will differ from using the optimal prediction? We assume that the estimator p' known to us is subject to error $p' = p + \varepsilon$, and that p' and α are large enough to $\hat{r} > 0$.

After inserting into (8) the $\hat{r}$ chosen according to (9) we get:

$$U(r, \alpha, p) = \log(1 + \alpha) - (1 - p) \log \alpha + p \log(p + \varepsilon) + (1 - p) \log(1 - p - \varepsilon)$$

The above case was largely simplified. In a real case we should choose a better utility function, develop an accurate predictive model for events, including dependence between events, and the predictive model should make use of new data during the process of betting.

– So what's the difference between you and me?

– Because you don't have to survive.
You're weak. You have emotions.
You play little games with your mind.
You chase your tales.

Charles Manson (interview)

3 The task of rating prediction

This chapter introduces the analyzed task, collaborative filtering prediction of movie ratings. Collaborative filtering methods are a way to personalize a service by selecting content to present based on guessed preferences of each user.

Because we are interested especially in the application in collaborative filtering recommender systems, first in section 3.1 I discuss various issues considered when trying to build a good quality recommender system. Particularly of interest to us is the role and forms of prediction algorithms that appear in these types of applications.

Next in section 3.2 I discuss the notion of collaborative filtering itself, its short history, and its current understanding as prediction of missing values in a sparse matrix. I discuss the best known collaborative filtering methods, and applications other that the use in recommender systems.

Section 3.3 presents the first summary of the analyzed Netflix Prize dataset, containing over 100 million movie ratings.

Section 3.4 discusses how to evaluate rating prediction, and how to evaluate recommender systems. The evaluation measure chosen for the Netflix dataset was RMSE (root mean squared error). I will outline, how to adapt the methods optimizing RMSE to calculate recommendations. I will list RMSE variants for different choice of hold-out sets, used in this and other works based on the Netflix dataset.

In section 3.5 I will explore further the Netflix Prize dataset, presenting basic summarizations, tables and plots, that will give the first perspective on the dataset, before proceeding to the description of collaborative filtering algorithms used on that data.

3.1 Recommender systems

The main point of focus of the work are collaborative filtering recommender systems, in particular, collaborative filtering methods with best possible predictive accuracy measured on the chosen large dataset. We evaluate the predictive accuracy by RMSE on a held-out test set. In this section I will outline what recommender systems are, what is their purpose, and briefly describe the most important issues to consider when trying to build a good quality recommender system. I will discuss, how relevant is the chosen prediction task as part of a larger context of developing a recommender system, and in which aspects the rating prediction methods need to be adjusted to produce good quality recommendations.

A recommender system is software that presents a user a list of personalized recommendations, prepared on the basis of guessed user preferences. Another definition from Wikipedia is: "Recommender systems, recommendation systems, recommendation engines, recommendation frameworks, recommendation platforms or simply recommender form or work from a specific type of information filtering system technique that attempts to recommend information items (movies, TV program/show/episode, video on demand, music, books, news, images, websites, scientific papers, etc.) that are likely to be of interest to the user".

For a user, employing a recommender system is one of the ways to reach the information that interests him. Discovering or rediscovering information is all the more difficult, as we have now a situation of information explosion – sudden appearance of large amounts of data, mainly human-generated content on the Internet. A user wants to find relevant, good quality information in these large datasets, and filter out irrelevant information, all while saving his time and effort. For this purpose usually standard techniques of information retrieval are used, like various types of search in conjunction with indexing content descriptions, metadata or tags. Recommender systems complement the information retrieval techniques in situations when the user does not know exactly what he is looking for, or when the content is not described accurately, or not popular enough to appear high on search results. Recommender systems gather information about likes and dislikes of each user, and utilize the mechanism of personalization. The gathered information about preferences and tastes that is expressed directly, for example by ratings, often turns out to be more informative and more important than the content-based information.

To be sure that we are solving the right problem in a recommender system, we have to look broadly on the human-computer interaction perspective – what is recommended to a user, how and why. Important is the statistical perspective: which data is collected and how is it collected, what is the meaning of the data, how to overcome various cold start problems, and what is the right prediction task to solve. Important is also the software engineering perspective: simplicity of design of the recommender system, reliability, speed, amount of resources used.

Many deployed recommender systems have been described in the literature [Gol92, Res94, Hil95, Kon97, Gol01, Sch02b, Mil03, Lin03, Ali04, Pra11, Ama12]. The article [Mon03] overviews 37 early recommender systems. At that time the most popular technique employed were recommendations based on user-user similarity. Currently increasingly frequent is the use of matrix factorizations. Other articles summarizing the subject of recommender systems are [Sch99, Ric10, Jan10].

Let's list the issues usually needed to be considered when developing a recommender system: (the list largely follows the presentations in [Her00a, Bur02, Her04, Sch07])

- Purpose of a recommendation system

  The most important point to understand is what is the goal of developing the recommender system. What is the benefit for people involved?

  There is a user viewpoint here: to easily and quickly find liked items, products or websites (the item can be any other information), save user's time, filter out irrelevant items. Depending on a specialization of a recommender system (on the type of items), construction of the recommender system may significantly differ.

  There is a viewpoint of the owner of the recommender system: to add value to the service, gain new users, increase sales, increase ad clicks, increase quality of leads sent. So a causal effect interests us – we look for a way to convince a user to perform an action. We could say that a recommender system plays a similar role to merchandising in retail. The article [Sch99] lists three ways in which recommender systems enhance e-commerce sales: turning visitors into buyers, cross-selling, and increasing loyalty. There may be also non-profit motivations.

  There is also a viewpoint of the owner of a recommender item: to advertise, attract interest to his own product, website, business, reach interested users.

- User's perspective.

  Let's look closer at one of the perspectives mentioned, the user's viewpoint. Here the design of human-computer interactions is especially important: how the user interface looks like? Is the interface simple and easy to use? How the recommender

is integrated with the larger service? What is recommended to the user and how? Which additional features are implemented?

Important parts of the design are navigation methods in the recommendation service, and interaction of recommendations with other techniques of information retrieval. Recommender systems are often complemented with features like: search, categorization, tags, specialized query interfaces [Gol92, Sch02b], faceted search, browsing similar items, social navigation, finding similar-minded users. The work [Sch02b] proposes meta-recommenders, which give the user extensive possibilities to customize a movie recommender system, allowing to filter movies on the genre attribute, MPAA rating, film length, objectionable content, critic's rating, or distance to theater, and allowing the user to indicate the importance of each filter. [Mey12] lists four core functtions of a recommender system: decide, compare, discover and explore.

Users may particularly like a service that minimizes their effort, for example, allows to rate whole groups of items (like movie genres), presents to the user a varied set of items (or groups of items) to rate, but a set of items known to the user. They may like a service that minimizes the number of ratings needed to obtain quality recommendations, allows to import or export ratings, or otherwise expressed preferences, between different services. Besides using ratings, a different idea is asking the user questions about his actual need, to let the user to express his preferences and get relevant, personalized recommendations quickly. Such decision-tree-based approach, thought as a decision-support process, was tried out by the Hunch.com website, where users answer series of questions on a selected topic, and at the end they are provided a recommendation.

Apart from presenting a list of recommendations or predicting user preference for a given item, one of the desired features is explaining recommendations to the user. Several different realizations of explaining recommendations were analyzed by a users study in [Her00b]. In that study the best accepted way of explanation turned out to be a three-bar histogram of ratings given to the movie by similar users. Other well performing explanations were similarity to other movies, and a favourite actor or actress.

An alternative is a black box approach, producing recommendations without further explanations. In fact, explaining to users the exact result of best matrix factorization algorithms would be a difficult task ([Pil09c, Pil09d] is an attempt). Fully explaining recommendations is a lot easier for item-item or user-user K-NN approaches, when the K parameter is small.

Different users can expect different properties of recommendations, as diversity, need for novelty, surprise factor.

One should consider, how the recommender systems are used, for example, common in commercial services is sharing a single account by multiple people. Another example – accurate recommendations seem to be more important for DVD movie rental than for movie streaming, where a user can sample a few movies before deciding on the one to watch [Ama12].

- Data collection

  Important questions to answer are which data to collect and how to do it? What does the data mean? How close is the meaning of the data to the desired meaning.

  Information about user preferences can be gathered in different ways, leading to datasets with different properties. Some kinds of data are more useful for recommendations, or easier to use properly than the others. We want the recommendation

algorithm to learn what user thinks about items he knows, and avoid learning the mechanism that presents the data to the user.

Preferences about items can be collected with active or passive user participation (called also explicit vs. implicit feedback). Explicitly (directly) expressed user preferences, such as item ratings, are more robust to the actual missing data mechanism than passively gathered user preferences.

Active participation is when the user expresses his preferences by conscious choice, using a specialized interface. The user can give a rating, usually on scale $1-5$ or $1-10$, or can click like or dislike buttons. Other types of explicit feedback can be: ordering items according to personal preference, choice of one item of two or more presented, adding an item to the list of favorites, to the queue to watch, or indicate that he does not know the item (has not seen the movie), or is not interested in the item. Collected feedback can be positive and negative, or only positive ("like" buttons). Having only positive feedback, it is desirable to augment it with information about which items were exposed (displayed, etc.) to each user. Some types of collected data indicate that a user likes an item, other indicate that a user needs an item, for example, purchase data [Pra11] or search queries indicate a need. More complex preference gathering processes are sometimes used, like conjoint analysis [Cha05] in marketing research, where a user chooses one product among a few in a specially designed survey. The work [Cha11] describes various experimental schemes of qualitative and quantitative preference elicitation. Users can indicate their tastes in different ways than by evaluating items, for example, it may be easier for a user to express that he prefers one item over another, or to indicate that he likes a certain actor, director or writer, or to answer a supporting question about what he is looking for. Recommendations can be also based on data gathered from personality quizzes [Hu10].

The second option is passive data collection (implicit feedback), where the preferences are not expressed by the user, but are inferred from user behavior. Passively gathered data are for example: click data, time spent on website, social network information. Usually passive data collection does not indicate accurately whether the user likes an item. Some types of implicit feedback are more accurate, for example: time spent on watching a movie, watching a video, etc.

Sometimes the influence of loopback effects on the operation of a recommender system needs to be considered. For example, if the recommender system is based on click data, a lot of clicks cause that the item is more often recommended, and more frequent recommendations cause even more clicks. The loopback effect may negatively affect the quality of recommendations, especially for passively gathered data.

Which way of collecting preferences is the best, this may differ depending on what kind of items are being recommended. The most common types of recommended content are: movies, music, books, games, places, websites.

In addition to data collected from users, a recommender system can make use of fixed data, such as metadata about items, tags, taxonomies, descriptions. For movies it may be genre, actors, writers, etc. We can also gather additional metadata about users: age, gender, location, language, etc. A recommender system that bases on metadata is called content-based. Often information about content is gathered by users in the form of tagging (called also social annotations, folksonomy). [Bur02] categorizes recommender systems into: collaborative, content-based, demographic, utility-based and knowledge-based.

The data analyzed in this work are movie ratings. It turned out, that ratings carry exceptionally large amount of information about the items, and for the purpose of recommendations it is usually it is better to have small number of ratings than to have a large amount of metadata [Pil09b]. Content-based recommendations are gaining importance in cold start situations, when there are only a few ratings available (for item or for user), for new users and new items, and for items from the "long tail".

One can reflect on the psychological perspective: why the user gives one rating and not another. The distribution of ratings varies between different recommender systems. On Netflix movies are rated on $1-5$ scale. The most frequent rating is 4, which appears in the Netflix Prize dataset about 32% times. On Youtube, on the same scale $1-5$ the rating 5 appeared over 90% times, and the second most frequent is rating 1 (Youtube data from 2009, before switching from ratings to binary assessments like vs. dislike). Probably that the main reason of the observed difference is that on Netflix the users rate movies to express their preferences to obtain better personalized recommendations. On Youtube the personalized recommendations module is not so exposed as on Netflix – it is not visible on the screen, where the video is rated. We can speculate that the main incentive behind giving ratings on Youtube was the intention to influence the visible average rating, to help the community in evaluating the video, hence extreme evaluations are prevalent there (incidentally, users were much more likely to give high ratings than low ratings). Displaying the predicted rating can bias the rating given by user, as shown by a user study in [Ado11], which supported the anchoring hypothesis: "Users receiving a recommendation biased to be higher will provide higher ratings than users receiving a recommendation biased to be lower". Besides the listed motivations to give ratings (improving the received personalized recommendations, influencing the displayed average vote, helping community), other motivations may be: fun factor, storing ratings as a help to the memory or to import them to other services, showing your taste to friends, expressing self [Her04].

It may be useful to separate the preference expression options in a user interface according to whether the user watched the movie or not (whether he knows or does not know the item). In the first case, the user can be presented options such as: give a rating or like vs. dislike, add to favourites, add to blacklist. If the user has not yet watched the movie, he needs a different selection of options, to indicate if he: wants to buy the movie now (or watch in a theater), wants to buy in the future, wants to watch for free (for example, in TV), is undecided (wants to be reminded about the item later), is not interested with the item, is not interested and does not want to recieve similar recommendations.

- Measure of accuracy

  To develop a recommenation algorithm we decide on a measure that evaluates quality of recommendations, and then we try to choose a possibly most accurate algorithm according to this measure. Multiple issues have influence on the perceived quality of recommendations. We can mention economical or psychological factors: user's satisfaction, ease of use, simplicity of the recommender system, opportunities to save user's time, ease of finding quality content. Guessing the current goal of the user may be important. Is it better to give a general recommendation according to the guessed user taste, or is it better to recommend an item similar to the being currently viewed, or to recommend items according to the last actions, like searches on the website and last viewed items? We can consider many other factors coming from each of the three coming from each of the three mentioned earlier perspectives: user's, site owner's (like increasing sales) and item owner's (like coverage – whether

the item has a chance to be recommended, even when it is new or niche, from the long tail). Factors listed above are difficult to measure. To gain more insight one could try comparing several versions of a recommender system using users surveys. A broad examination of multi-factor evaluation criteria with user surveys was performed in [Mee09, New10]. To simplify the problem and the resulting algorithms, we should narrow the accuracy criterion, and measure only the most important characteristics.

Selecting items to a top-K recommendation list is a classification task, and we definitely should examine, typically considered in classification tasks, precision-recall balance. Precision is the proportion of relevant items on the top-K list, and recall is the proportion of all relevant items retrieved to the top-K list (relevant items are the items that the user would rate high). An obstacle in measuring precision and recall on the Netflix dataset is that we have only ratings for user-selected items, and we do not have a good way to estimate which of all items are relevant for a user. The subject of precision and recall, and its relation to RMSE evaluation, missing data structure, and uncertainties in predictions, is expanded in section 3.4 "Evaluation". Typically used ranking-based criteria defined on the observed data (such as NDCG - normalized discounted cumulative gain, mean average precision, half-life utility [Her04], or probabilistic ranking cost functions [Bur05]) have the same disadvantage as mentioned top-K precision or recall – they ignore missing data effects, and hence do not evaluate accurately the error made on all items.

Ranking-based criteria are difficult to directly optimize. A well working simplification is to use an indirect, two-step method: first to predict user preferences (ratings) for unknown or unrated items, and then to sort items according to the chosen way of sorting predictions (to optimize a chosen ranking-based measure). Netflix proposed evaluating rating prediction by RMSE on the held-out set of newest ratings, and this criterion is used to evaluate methods in this work. In place of RMSE sometimes MAE (mean average error) is used in the literature. A disadvantage of using RMSE this way is that if we learn our algorithm on the training data distribution, predictions for missing data will be biased – if we asked the user, his rating typically would be lower than the calculated prediction. Also, RMSE allows us to assess if we predict the expected rating well (on the observed data distribution), but to calculate recommendations, our algorithm should also estimate uncertainty of predictions made (more about it in sections 3.4 and 4.2.4).

Overally, in this work I stay with developing methods to minimize RMSE, but to produce recommendations I correct the output of the methods (expected rating) by a rough estimate of standard deviation of predictions, and by an adjustment for the missing data structure (see section 3.4, and chapters 6 and 7). Users expect diverse recommendations, and rewarding diversity can be built-in into a ranking-based accuracy measure, or the calculated lists of recommendations can be postprocessed to provide more diversity.

- Diversity.

  Redundancy on a recommendation list is undesirable, as well as the list being too monotonous (a term sometimes used is serendipity – ability to pleasantly surprise the user). Recommendations serve partially as a discovery tool to help satisfy user's curiosity. Diverse recommendations increase perceived coverage, give an opportunity to explore the space of items. Diversity makes recommendations less limited to the group of items for which the user expressed his preferences earlier. If a recommender system makes a mistake by proposing an item not liked by the user, it is undesirable to have recommended similar items, for which it is likely that we have made the same

mistake. In other words, we do not want the errors for items on a recommendation list to be correlated.

If several very similar movies have a high predicted score, for example, when there are several versions of the same movie in the system, or unintended duplicates in the database (frequent case in practice), then it is better to not occupy the recommendation list with all of those similar movies, but to choose one or two of them. The usual sorting by expected rating corrected for uncertainty does not ensure diversity. Different ways are possible to increase diversity at the cost of probability of making a correct recommendation (accuracy) [Kwo11]. The sorting can be improved by removing similar items from the recommendation list. A solution described in section 4.5 was to recommend whole clusters of similar movies. Different approaches to ensuring diversity can be compared, for example by using measures as intra-list similarity [Zie05], "personalization" [Zho10a], "surprisal" [Zho10a], or unexpectedness [Ada11] (details in the respective papers). An untested criterion rewarding diversity to try out is to maximize the probability that at least one item on the top-K recommendation list will be rated higher than a given, sufficiently low threshold (evaluating the whole lists, taking into account dependencies between ratings of different items).

A related problem to ensuring diversity is detecting similar or equivalent items and removing them from the recommendation list. It is obvious, that if e.g. the user rated (implying he probably viewed) the movie, has read the book, bought the product, then the same items or equivalent items should not be recommended to him, like another version of the same movie, another edition of the book, an identical product by another producer. Additional rules are, for example, that a movie sequel should not be recommended before watching the first movie, different seasons of the same TV series should be recommended in sequence, etc. Detecting equivalent items is a domain-dependent issue. It is a separate machine learning task and the automatic solutions may be not obvious.

It often happens that a user does not know what to do with a recommended item entirely unknown to him, or is undecided about an item. Some solution to that is ensuring temporal diversity [Lat10], that is, besides ensuring diversity inside a single recommendation list, changing the recommendations over time.

- Balanced recommendations: popularity vs. long tail. Cold start issues.

  Collected user preference data usually has a "long tail" structure. Items from a small group are much more frequently rated than the rest, and a small group of users rates much more items than the rest of users. For example, in the Netflix Prize training set the most popular 10% of movies received over 76% of all ratings, and the 10% most frequent users gave 44% of all ratings. In long tail situations the distribution of objects often follows a power law, that means the frequency $p(n)$ of an object with the frequency rank n in the database is proportional to n^{-a} for some constant a. In the Netflix Prize data the frequency of movie ratings is approximately distributed according to $p(n) \approx \exp^{-a\sqrt{n}}$ for different movie frequency ranks n, and the user frequencies are distributed approximately log-normal, so neither of the two distributions is of power law type. Section 3.5 expands this subject. The paper [Bry07] examines the long tail in product sales over the Internet, and the papers [Tan09, Goe10] examine the long tail in the Netflix dataset.

  Recommendations need to be balanced between recommending popular items and recommending the long tail of less popular items. It is easiest to recommend proven, good quality popular content, because for it we get accurate predictions. One reasonable option is skipping the long tail entirely, as, for example, I have done in the

application described in section 6.2.3, which recommends only the most frequent 2000 movies from the Netflix dataset. Usually we need to recommend the long tail to some extent, for example, new items with few ratings need to be recommended, so that they have a chance to eventually become popular. We can mark out the notion of coverage of a recommender system – whether all items have a chance to be recommended, and how often are they recommended. Similarly, there is a need to balance between familiar content, that the user will like with high certainty, and the content unfamiliar for the user, for which there is larger risk of making a mistake in recommendation. One way to balance the recommendations is to tune the amount of penalty for the variance of a rating prediction. Decreasing the penalty causes recommending more rarely rated content with uncertain predicted ratings: new items, or relatively unknown or niche items. Users can have different individual preferences for taking risks (or preferences, for example, for the surprise factor), and there may be a need to adjust the balance individually for each user. This is similar to the risk vs. reward relationship in finance.

New items go through a phase of having few ratings. Similarly, new users initially do not have any ratings, and the items recommended first may be completely unknown for them. Those cases are called cold start problems for users [Sch02a] or items. We can also speak about a global cold start problem for the whole recommender system. One idea to overcome the cold start problems is to use content-based recommendations in a case, when there are few ratings. The content used can be metadata about items, like, for example, for movies it will be year of production, actors, writer, director, genre of the movie. For users the metadata can be age, gender, location, etc. A recommender system may encourage to rate rare items (it would be best to recognize which rare items have higher chance to be known by the user). Content-based recommendations were used for items with no ratings in the application described in section 6.2.3, and in section 6.1 we propose heuristic solutions for cold start situations.

In overcoming the cold start problem for users a specialized interface can help (so it is done in the Netflix service), encouraging new users to quickly rate a number of popular items, and allowing to express preferences about large groups of items, like the whole genres.

- Choice of algorithm

  After deciding on a criterion of evaluating recommendations remains the choice of a best performing algorithm on the given criterion, for available data.

  The choice of hold-out RMSE for evaluation by Netflix made developing machine learning algorithms convenient, because it skips the usual non-trivial issue of domain adaptation in real-life tasks, when the gathered data has a different distribution than the data needed at the moment of usage. Another advantage is that using MSE loss and approximating ratings with Gaussian distributions allows us to employ standard linear models, like regularized linear regression. Improving accuracy of rating prediction translates well to better realizing other goals such as better recommendations, better item-item similarities, clustering, visualization, better user-user similarities. To obtain good accuracy, we need to process a large collection of data, that is why we have also pay attention to computational aspects of the method, as time complexity and resources used.

  We can observe, that no matter what kind of data about user preferences we have gathered, most accurate algorithms for recommendations seem to always contain dimensionality reduction. For prediction of movie ratings on the Netflix dataset, as

evaluated by hold-out RMSE, most accurate were variants of sparse matrix factorization (regularized SVD) using appropriate regularization. Techniques of prediction of movie ratings are shortly summarized in section 3.2 "Collaborative filtering", and thoroughly discussed in chapter 4. Chapter 6 describes an example use of regularized SVD in two recommender systems.

- Temporal effects.

  The basis of why collaborative filtering methods work is the assumption of item persistence and user taste persistence, meaning that the properties of items and the user preferences for items do not change by much, even during a longer period of time. One might wonder, how exactly the model of data changes in time, for example, how much does a rating made several years ago say about the present preferences of a user. The evolution in time may differ for different types of items. For items like movies, books, music, the model will likely change slower than for items that are altered over time, like websites, or for items with short-term usefulness, like news. Modelling the variability over time can significantly improve the prediction accuracy.

  In the Netflix dataset of movie ratings, temporal effects improved RMSE accuracy of all groups of methods [Bel07c, Kor09a, Kor09b, Tos09, Pio09]: matrix factorizations, RBM, K-NN, kernel methods. The effect that improved accuracy the most was the single day user bias [Tom07, Pot08, Bel08] (the large single-day effect may be partially a result of using single Netflix accounts by multiple persons). Other useful effects were short-term and long-term variations in user biases, user preferences and movie biases, and modelling the so-called frequency effects [Pio09]. As for the computational complexity, matrix factorization models with user preferences changing in time have many times more parameters than the regular matrix factorizations, but the emerging models can be efficiently learned with gradient descent [Bel08, Kor09a, Kor09b]. Easier is enhancing the distance-based methods, such as item-item K-NN, by correcting the distances between items for difference of rating dates [Tos08b, Tos09] (see also section 4.5.1).

  Single day bias improves predictive accuracy measured by RMSE, but in applications in recommender systems it may be not that important, because the bias is equal for all items at the time of calculating recommendations. The day effect on user bias does not directly influence the ordering of items, but its presence in the model can alter other model parameters and thus slightly affect the quality of recommendations.

  More important are short-term changes in user preferences. Ratings or likes and dislikes are not a good way to indicate those short-term changes, and we should take into account the current context of use of a service. To learn about user's temporary preferences it is better to rely on data such as the recent search query, clicked tag, visited item page, or to give a user an opportunity to indicate his current mood (for example, a preferred genre, or a genre to avoid).

  To explain the nature of short-term and long-term temporal effects a psychological perspective may be needed – why the user gives one rating over another. Some psychological aspects of movie evaluation were considered in [Pot08].

  Another issue is the need of temporal diversity of recommendations, already mentioned when we discussed diversity.

  We can also verify if recommendations are stable [Ado10], that is whether predictions do not change much over time when the user gives ratings agreeing with the previous predictions made for him by the algorithm (note that the predictions made for all items should depend on the missing data structure).

- Computational perspective.

  A working recommender system can potentially serve millions of users, recommending possibly hundreds of thousands of items, and processing billions of ratings.

  For this scale of volume of data processed special attention must be paid to how much resources the recommender system uses, such as: the number of computers realizing the recommender service, the number of running processes, the amount of memory used, RAM and on hard-drives, the amount of network bandwidth occupied. We may wonder how the amount of resources used scales with a growing number of items, and with a growing number of users.

  The perceived speed can be important for users. This leads to questions about the recommender system: are the recommendations instant – is the new list of recommendations available imediately after rating the next item? How many lists of recommendations the system can generate per second? How many items are on one computed list of recommendations?

  For new items sometimes it would be useful to use new ratings quickly to accurately recommend this new item. We could inspect what is the latency of updating information about items – the information such as the movie variables in the matrix factorization, or the distance between items in the K-NN algorithm.

  Different versions of algorithms have different speed and resource usage. Algorithms based on matrix factorization are usually faster than those based on K-NN.

  Additionally, specialized caching may be used, to avoid calculating the same recommendation lists multiple times.

- Software engineering perspective.

  A recommender system is a piece of computer software that is required to operate online for an extended period of time. Developing a recommender system is in a large part a software engineering task, and one needs to consider issues like simplicity of construction, whether the system is easy to expand, easy to implement new features, to test, to create and recover backups. Is the system built in a way that supports assuring reliability? What administrative and monitoring tools were developed? What is the anticipated uptime? What is the typical and the maximal consumption of resources?

- Security.

  As any software, recommender systems need to be examined with regard to security. Besides securing against common threats such as data leaks or denial of service attacks, typical risks for recommender systems are shilling attacks [Lam04, Meh09], and risks to users' privacy.

  A shilling attack means artificially influencing item position on recommendation lists, by adding votes with synthetic profiles. Shilling attacks may be performed by users who like or dislike an item, or by content owners who want to increase exposure of their content or to put down competitive content. Such rating manipulations are undesirable, because when ratings does not reflect the real preferences of users, the quality of served recommendations can decrease (some call resistance to shilling attacks as "trust" [Sch07] of a recommender system, but the notion of trust is also used to describe social type of recommendations using opinions of friends or selected people who we trust about their good taste [Ric10]). Shilling attacks can to some extent be prevented by identifying malicious profiles through examining IP addresses or browser fingerprints, or a larger weight can be put in the collaborative filtering

algorithm to votes of committed users, who gave many votes, made a purchase, wrote reviews, earlier registered on the website, etc. Also, machine learning techniques can be employed to identify atypical patterns in data.

Ratings in a recommender system can be publicly available or anonymous. If they are anonymous, the issue of privacy emerges – determining if there is a possibility of disclosing the ratings, and what may be the consequences of disclosure. Probably there is no need to assure maximum possible privacy in recommender systems, as it is in case of, for example, storing credit card data, or ensuring security of bank accounts. If disclosing ratings of a very small fraction of users is possible, it rather is not a factor disqualifying a recommender system. Some consider algorithms ensuring differential privacy [McS09], meaning a setting when adding a new rating to a recommender system does not allow to infer information about other ratings in the system. The feature of differential privacy was not considered in the methods developed in this work.

There was also discussion of privacy risks of the published Netflix Prize dataset. Users from the anonymized Netflix dataset can be correlated with publicly available ratings in other services, and as an example [Nar08] correlated two users from the Netflix Prize data with authors of rated IMDb reviews. It is a good case to think on privacy issues in recommender systems, and what happens when it is possible to obtain ratings for some subset of users. In rare cases a user can have reasons to rate publicly a fraction of movies he watched, but hide his ratings for some other movies. To talk about a privacy breach or de-anonymization, many conditions have to be met. First, a large number of users with publicly available ratings in an other database should be identified in the Netflix database. Matching users is inaccurate many reasons: there is inherent noise in data (user's taste and mood change over time, and re-rating experiments show [Ama09], that a user often gives different ratings to the same movie), the Netflix Prize data was additionally perturbed, websites often use different rating scales ($1-5$ for Netflix, $1-10$ on IMDb), and the distribution of ratings differs between websites. Because matching two large databases of users is a multiple comparison task, much larger confidence is needed to determine if two users are the same than in a case of comparing two users only. Even if a user is with certainty identified in the Netflix database, because the user already showed part of his ratings publicly, and because of all uncertainty with rating perturbation, ratings changing after re-rating, tastes changing over time, and the possibility giving a rating by mistake, it seems to be unlikely, that large fraction of users would object to publishing their Netflix ratings. The selection of movies in the Netflix Prize data rather does not contain highly controversial titles, for which users would have a really good reason to hide ratings. Of course, as we will see in section 4.3.4, ratings tell a lot about psychological profile of a user, so privacy concerns about releasing any amount of ratings are justified.

The recently released Million Songs Dataset [McF12] contains song playcounts for each anonymized user, but does not contain the song timestamps – the creators of the dataset call the associated privacy risk "limited and reasonable".

I have listed some of the most frequent issues considered when designing and developing recommender systems. The rest of the work is focused mainly on rating prediction evaluated by hold-out RMSE. Chapter 6 describes the adaptation and deployment of the developed prediction methods in two recommender systems.

3.2 Collaborative filtering

What is collaborative filtering? A short definition from Wikipedia is: "Collaborative filtering (CF) is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.". The most common use is predicting users' preferences for items, used in recommender systems, that is why collaborative filtering is often defined in the context of applying in a recommender system: "The collaborative filtering technique matches people with similar interests and then makes recommendations on this basis" [Mon03].

The idea behind collaborative filtering methods is intuitively obvious: there are groups of users with similar tastes, meaning that they rate items similarly, so when we want to recommend items to a user, items can be filtered through the tastes of similar users. The concept of collaborative filtering evolved during its short history around the initial idea of user-user similarities. One of the first applications was filtering a mailing list by multiple moderators [Gol92], explicitly chosen by the user. Then appeared applications based on gathering preferences, like ratings [Gol01], where users with similar taste were found automatically and recommended was the content liked by the similar users. Most of the early recommender systems were based on user-user K-nearest neighbors prediction [Mon03]. User-user similarities have an additional advantage of being intuitively understood, as, for example, statistics summarizing how similar users rate an item are a well understood explanation why the item was recommended [Her00b].

Further scrutinizing the collaborative filtering idea, the gathered user preference data can be used, besides identifying similar users, to identify similar items. On the Netflix Prize task item-item K-NN methods were more accurate than user-user K-NN. Item-item similarities also allow to create local recommendations of type "if you liked this, you will probably like that", related to the item, which the user is currently interested in. Because similar users tend to give near ratings to groups of similar items, we can predict unexpressed user preference for an item, even when similar users have not rated the given item.

The present understanding of the crux of the task is that collaborative filtering boils down to predicting missing data in a sparse matrix. In this sense, collaborative filtering is not just about recommendations and filtering content, but one can also speak about applying collaborative filtering in any domain where there is a need of matrix completion (partially or fully), for example, collaborative filtering was used for medical datasets [Has10] or educational data [Tos10].

For prediction of item ratings, the most accurate collaborative filtering methods found, such as matrix factorization methods, base on dimensionality reduction. Intuitively, dimensionality reduction methods can be understood as automatically learning hidden genres of items and learning users' preferences for these genres. [Ski07] lists the following interpretations of matrix decompositions: factor interpretation as signals from hidden sources, geometric interpretation as hidden clusters, component interpretation as underlying processes and graph interpretation as hidden connections.

For the data and task in this work, the best matrix factorization methods found were more accurate with similar computational requirements than the best near-neighbor methods. In contrast to K-NN based methods, in methods like matrix factorization we can say that to predict one particular rating for one item and one user the whole rating matrix is used – both similar users, and not-similar, both the movies he likes, and the movies he does not like, and all movies liked or disliked by all other users.

Due to non-intuitiveness of the automatically learned genres, matrix factorization based recommender systems are usually black-box-type, without explanation (or pseudo-white-box, with partial explanation), while K-NN methods allow for intuitive, white-box explanation of prediction values when the K parameter is small.

A useful framework encompassing the most accurate families of collaborative filtering methods was multitask learning [Yu03, Bak03, Car97], where tasks are users (rows of a sparse matrix), and task attributes are items (matrix columns). Similarities between tasks and similarities between task attributes are captured using a hidden structure of parameters, usually a set of parameters individual per task (user) and a set of parameters describing task attributes, shared by all tasks.

The central theme of this work are accurate methods of collaborative filtering. The content of the work and the taken approaches were determined by the chosen task: prediction of held-out movie ratings in the Netflix Prize dataset, which contains over 100 million ratings on scale $1-5$, made by 480189 users who rated 17770 movies. An advantage of the chosen dataset and the task of minimizing RMSE was its popularity, and in the work I hope to summarize the present understanding of the problem and developments that resulted from efforts of many people. As we will see further, for a real-life task relatively many problems of statistical or mathematical nature appeared on the way.

It is difficult to identify precisely the model from which data come from, that is why the best approaches were ensembles of many accurate, yet differing methods [Tak07a, Wu07, Pat07, Bel07c, Bel08, Kor09b, Tos09, Pio09]. It should be noted, though, that in a practical application to achieve satisfactory predictive accuracy it should be sufficient to use one method.

The most accurate single methods calculated low rank matrix factorization with proper regularization [Fun06, Har07, Sal07a, Bel07c, Tak07a, Lim07, Tom07, Sal08], usually using 30 to 200 hidden item and user features. Well performing were also appropriately tuned variants of K-NN method [Tos08b], kernel methods [Pat07, Yu09a], and Restricted Boltzmann Machines [Sal07a]. Accuracy was improved by enhancing models by using implicit information [Sal07a, Pat07, Sal07b, Bel07e], time information [Tom07, Bel08, Kor09a, Kor09b, Tos09, Pio09, Xia09] and integrating models with K-NN [Bel08, Kor09b, Pio09].

3.3 The dataset

Most experiments listed in this work were done on the Netflix Prize dataset. I will outline, what data are included in this dataset, and what the accompanying prediction contest was about, which gathered large interest.

The Netflix Prize dataset, published in the year 2006, contains the files: training.txt, probe.txt and qualifying.txt. The file training.txt contains $100,480,507$ records (user_id, movie_id, rating, date). The ratings, on the integer scale $1-5$, were gathered during the period from October 1998 to December 2005. The data was additionally perturbed by Netflix by modifying ratings and dates, and deleting ratings. The perturbation made did not change the RMSE score of Netflix Cinematch [Ben07]. Ratings were given by $N = 480,189$ users (anonymous user ID's), who rated $M = 17,770$ movies. If we treat the data as a sparse matrix with $N * M$ elements, about 98.9% of ratings in the matrix are missing.

The file probe.txt contains $1,408,395$ records (user_id, movie_id), for which the corresponding ratings are present in the training.txt file.

The file qualifying.txt contains $2,817,131$ records (user_id, movie_id, date). The corresponding ratings for these records were published after the end of the Netflix Prize competition in 2009. The task of Netflix Prize was to propose point predictions for ratings, that minimize the RMSE (root mean squared error) criterion on the qualifying set.

The data in qualifying.txt and probe.txt are from the same distribution – a subset of latest ratings for each user (maximum 9 ratings of one user in total) was split randomly in a 2:1 ratio between the qualifying and probe sets. The qualifying set was further split in a 1:1 ratio between the quiz and test set. RMSE on the quiz set was possible to check by the automatic evaluation system by Netflix. The undisclosed earlier RMSE on the

test set was the final criterion of evaluating solutions after the end of the competition. Submitted predictions were compared with the results of the reference algorithm Netflix Cinematch, which had $\text{RMSE}_{quiz} = 0.9514$ and $\text{RMSE}_{test} = 0.9525$. The Netflix Prize task was to develop an algorithm better than 10% than the Netflix Cinematch, that is to reach $\text{RMSE}_{test} < 0.8572 = 90\% * 0.9225$. In the next section "Evaluation" more is said about the choice of RMSE as an evaluation measure, and about different RMSE's for different held-out test sets in the Netflix Prize dataset.

The Netflix Prize contest was launched in October 2006, and continued until June 2009, when two teams crossed the 10% threshold. It was won by a 7-person team Bellkor's Pragmatic Chaos, and in accordance with the rules of the contest they described their solution in the set of reports [Kor09b, Tos09, Pio09]. Parts of the final solution are described in the previous Progress Prize reports [Bel07c, Bel08, Tos08b]. Over 5,000 teams submitted their solutions to the contest. The solution of the author of this work was in final place 43rd, with score $\text{RMSE}_{test} = 0.8717$, that is 8.48% better than Netflix Cinematch.

In addition to the training, probe and qualifying sets, the Netflix Prize data includes the file movie_titles.txt, containing the movie titles and the years of release. There were attempts to use those data for prediction [Tak07a, Tos08b], but there was no accuracy improvement, or the improvement was minimal. [Tos08b] used the release year, and [Tak07a] tried using similarities between titles. Also, there were many attempts to use additional metadata about items, besides the release year using features such as: genres, actors, writer, director, gathered from datasets like IMDb [Pil09b], or Wikipedia [Lee08], but it turned out that ratings contain enough information, and additional information about movies does not improve prediction [Pil09b]. I have also unsuccessfully tried to improve prediction accuracy in the Netflix task by using metadata, though I used IMDb metadata to predict movie features for new movies (see sections 4.7 and 6.2.3).

The intention of focusing on one dataset and task is understanding the dataset well and developing possibly most effective prediction algorithms. A plus of the choice of the dataset and task of Netflix Prize is large interest among professionals, which ensured good quality of developed solutions. An advantage is also that it is a large collection of data, at the time of writing this work it is the second largest publicly released set of item ratings (the largest one being the KDD Cup'11 Yahoo! dataset with over 300M ratings for over 600K items). Other popular datasets of movie ratings are EachMovie, containing 2,811,983 ratings and MovieLens containing currently three sets: 100K, 1M and 10M ratings.

About the engineering matters, doing calculations on data of size 100 million ratings is on boundary of the capability of present-day personal computers, and an adequate representation of data was important. Using a traditional SQL database would be too slow, that is why the role of a database was fulfilled by appropriate C++ data structures. I decided on using a simple structure, an array with 100 million 3-byte records – one byte for a rating, two bytes for a movie ID:

```
struct Data {char rating; short movie_id;} __((packed));
```

or additional two bytes for the day when the rating was made:

```
struct Data {char rating; short movie_id; short day;} __((packed));
```

The array is indexed by additional arrays, pointing where each user's ratings are placed. Users and movies were sorted according to the number of their ratings.

To avoid parsing the data each time an algorithm was started, the array was allocated with a mechanism of mapping a file to memory (mmap function of the Linux operating system). This data structure was good enough for experiments. The memory occupancy was about 300MB in the first version and about 500MB in the second version containing the date information. The algorithms described in this work most often took 5-15 hours

on average PC, for learning parameters and computing predictions. Because the majority of algorithms were iterative and they had to repeatedly iterate reading data, some teams experimented with compressing the data structure [Tak07a], which allowed to further accelerate the algorithms.

A few times in the work I used a small subset of the Netflix Prize data, that was more dense, to be suitable for testing algorithms that use missing data imputation. For the dense dataset 500 most popular movies were selected, and 10059 users who rated at least 250 from the 500 most popular movies, but rated at least 285 from the remaining movies. This criterion was chosen to filter out users, who rate too many movies, supposedly rating many movies they had not seen, and to counter overrepresentation of users with mass taste (users who like popular movies). The resulting dataset is a matrix 10059×500 of ratings, which is over 56% dense, in comparison to the 1.1% dense whole Netflix Prize dataset. Methods were compared using the following evaluation: $304,500$ ratings were randomly selected to the test set, so that the remaining training set contained exactly $10059 * 500/2$ ratings, being exactly 50% sparse. RMSE calculated on a random draw of the test set (the same test set for all evaluated algorithms) is called RMSE_{small} (all RMSE versions used in the work are listed in the next section 3.4 "Evaluation"). Evaluation by RMSE_{small} could be improved by using an cross-validation-like algorithm, if needed, by averaging over several draws of the test set.

3.4 Evaluation

The previous section introduced the Netflix Prize dataset containing the files training.txt, probe.txt, qualifying.txt and movie_titles.txt. As a training set in my experiments the data from training.txt was used with 15% of probe.txt excluded. On this training set each implemented method was trained once. The excluded portion of probe.txt was used as a test set to evaluate methods, tune parameters of the methods, blend predictions, and to observe how well different kinds of methods combine with each other (and draw conclusions from that).

The task proposed by Netflix was to predict ratings from the test set distribution (probe.txt and qualifying.txt), and evaluate predictions by comparing root mean squared error (RMSE):

$$RMSE(\hat{y}) = \frac{1}{|Te|}\sqrt{\sum_{ij \in Te}(r_{ij} - \hat{r}_{ij})^2}$$

where Te is a chosen hold-out set, that is a subset removed from the data while training (later we list different hold-out sets used). The probe and qualifying sets consisted of up to 10 most recent ratings of all users (for most users exactly 10 ratings), so they have a different distribution than the training set, but in experiments there was close to no difference between using the data from the probe set or from the training set for hold-out evaluation.

I will discuss advantages and imprecisions of the choice of RMSE minimization on a hold-out set, from a perspective of using the resulting rating predictions in a recommender system. Optimizing RMSE usually results in simplest and fastest algorithms, in comparison to other possible criteria evaluating prediction of ratings, or criteria evaluating personalized rankings of items directly [Wei07]. For example, if we want to approximate a set of values with one number, minimizing RMSE will give the average, and minimizing MAE (mean average error) will give the median, which is more complicated to compute.

The task of calculating recommendations is different from predicting ratings, but, as argued later in the work (sections 4.2.4, 6.1 and 6.2.3), the methods predicting ratings, developed to minimize RMSE on the held-out observed data, can be adapted to obtain good quality recommendations (for example, it is certain, that to generate recommen-

dations we need to adjust the predicted expected ratings by uncertainty of predictions). Let's look closer at how rating prediction on the observed data distribution is related to the task of calculating recommendations, which can be seen as a classification task – to identify relevant items. For example, accurate estimation of user bias (or user mean) is important when we predict ratings, but when we calculate recommendations, user bias is the same for all compared items, on the other hand, item bias is very important for calculating recommendations (and also, is prone to missing data effects). Perhaps we can find a better criterion to optimize methods than RMSE, and find better criteria to evaluate recommendations than the typically used ranking-based criteria calculated on ratings from the observed data distribution, as for example, top-K precision, NDCG (normalized discounted cumulative gain), etc.

Let's assume that we have a method that, for a given user and item, outputs one value, the predicted rating. Let's assume that the user gives a rating 5 for items that are relevant for him, and we want to identify the unrated relevant items to put them on the top-K list of recommendations for the user. We can define a loss function that weights different kinds of errors made by our prediction method. The following table lists my estimation of how large the cost of error should be for several possible cases:

Predicted rating	Actual rating	Cost of error
5	1	10
5	3	3
1	5	1
1	3	0

If the algorithm inaccurately predicts 5 when the real rating is 1, it will cause us to make a bad recommendation, likely leading to a negative experience for the user, hence the cost of such case should be large. If we inaccurately predict 1 when the real rating is 5, we are just skipping a relevant movie, without a negative experience for the user (but some users may accept more risk if it leads to a better chance of discovering a relevant movie). We can say, in an intuitive sense, that precision is much more important than recall. Perhaps, to simplify, we could skip entirely evaluating recall, and evaluate only precision (if we only consider relevant vs. irrelevant classification, precision and recall will be equivalent for a fixed K). Looking at the table, it seems that evaluating the calculated top-K lists of recommendations by $\frac{1}{K}\sum_{i \in top-K}(5 - r_i)^2$ could work well – let's call this measure MSE@top-K. In practice we want diverse recommendations over time, and we can vary the K parameter, and select items for recommendation from a larger set, increasing the weight on recall. Typically, K is small, and in consequence we feel that if we skip a relevant item on the recommendation list, it can always be replaced by a slightly less relevant one, but in reality we should also reward recall – recommendations should be diverse, and the interface of the recommender system should provide options like search, categorization, meta-data navigation, rating groups of items, tagging, to give the user opportunities to visit and rate items outside of the preferences declared so far.

Above we considered methods predicting the expected rating, but we can do better with methods that additionally are able to assess the uncertainty of predictions made. Generally, for movies with more ratings we get more accurate rating prediction than for movies with few ratings. This fact can be used to improve recommendations. While predicting ratings, the appearing local posterior distributions of ratings can be approximated well by Gaussians $N(\mu, \sigma^2)$, clipped to range $[1, 5]$, and, if necessary, rounded (see section 4.2.2). Having the estimated posteriors $N(\mu_i, \sigma_i^2)$ for each movie i, in order to minimize expected MSE@top-K we should choose to the top-K list the movies with the highest score function: $s(\mu_i, \sigma_i) = \int_1^5 (5-x)^2 N(\mu_i, \sigma_i^2) dx + \Phi((1-\mu_i)/\sigma_i)$. Because we already made several large simplifications in the above considerations, and the whole reasoning is inaccurate, it is all

right to further simplify the above expression. In this work I decided to score items for recommendations by $s(\mu_i, \sigma_i) = \mu_i - C\sigma_i$. Because selecting K items from a larger set is a situation of multiple comparisons, the constant C increases with the number of items considered. With a typical "long tail" distribution of item ratings there are few items with low σ_i and many items with high σ_i.

We know more or less how to produce recommendations from predictions for all items. The hatch is that the methods learned on the training set produce accurate posterior predictions for the training set distribution (items likely rated by the user), but predictions for all items (items selected for rating uniformly at random) typically are largely biased. Obtaining accurate posterior distributions for all items is a domain adaptation task, for which we need additional data where users are forced to rate randomly selected movies (ideally with an additional option to indicate that the user has not watched the movie, instead of giving a rating). The Netflix Prize dataset does not contain additional non-user-selected data, as it is provided, for example, in the Yahoo! Music R3 dataset, and I rely here on using heuristics to correct the methods trained on ratings for user-selected movies only. The proposed heuristics (section 6.2.3) penalize distance (dissimilarity) from the set of movies rated highly by the user. Logically, if this distance increases, the mean μ_i should decrease, and σ_i should increase, but a more precise relationship and the best distance to use need to be determined on additional data (rated, randomly selected movies). Note that the structure of missing data may originate from various causes unrelated to user preferences, for example, it can depend on which items are exposed by the website navigation or by the interface of the recommendation service. In consequence, the relationships between the probability of the rating being missing, and the parameters μ_i and σ_i, can differ for different groups of items.

To further illustrate the problem of using data for recommendations let's look at a simplified case. Assume that the set of items for a typical user in a typical recommender system can be divided into five parts: R5, R1, U5a, U5b, U1. By R5 we denote the set of items rated 5 by the user (let's say, about 100 ratings). By R1 we denote the set of items rated 1, which typically is smaller than R5 (let's say, about 10 ratings). By U1 we denote missing data, which in reality would be rated 1 or marked as not known to the user (about 10,000 ratings). By U5a we denote missing items, that would be rated 5, and are similar to the set of items rated by the user (about 100 ratings). By U5b we denote missing items that would be rated 5 (after watching, in the case of movies), and are not similar to the set of rated by the user, but are from regions (of the universe of items) unknown to the user (about 1000 ratings). Now, with this division, the recommendation task can be defined as predicting in the missing data set (U5a,U5b,U1), which items belong to U5a (if the user expects "safe" recommendations, which he can immediately evaluate), or which belong to U5a+U5b (if the user is interested also in content unfamiliar to him, which he needs to examine before evaluating, watch the movie, etc.). Note that if users are forced to rate unfamiliar content, as it is the case of the dataset Yahoo! Webscope Music, the gathered ratings will be lowered comparing to how users would rate the items after they familiarize with them. We can expect, that the algorithms described in this work (minimizing RMSE for held-out ratings from the data distribution) will predict a low rating anyway for the items in the U1 group, so items from U1 are unlikely to enter the calculated Top-K lists of items. Experiments on datasets with MCAR data (missing completely at random; non-user-selected) are needed to confirm or deny this guess. If the missing data influence has the form of a systematic negative bias, which affects all missing ratings from the U1 group in a similar way, taking it into account will not impact recommendations largely. If, however, the negative bias largely differs between groups of items, it should be modelled to improve the quality of recommendations. Similarly, as we mentioned earlier, we should also estimate changes in uncertainty of predictions. Another question is, how important

for quality of recommendations is distinguishing U5a from U5b.

In [Mar05] models CPT-v and LOGIT-vd were proposed, which correct dimensionality reduction methods by additional global biases, that adjust the models to predict ratings that do not come from the data distribution. Those methods need for learning an additional small amount of rated non-user selected items. [Mar08] also proposed a method cRBM/E-v, which modifies Conditional RBM [Sal07a] by adding five global biases active for non-user selected items, one for each value of a rating. As experiments show [Mar08, Mar09], the biases used in methods cRBM/E-v, CPT-v, LOGIT-vd to a large degree correct the underlying algorithm trained on user selected data, so that it has good prediction accuracy also on non-user selected data (such as uniformly missing items). The correction by simple global biases should not influence much the relative ordering of items, but it is possible, that a more complex and accurate model of correcting for the situation of items uniformly missing would have impact on recommendations. An obvious possible extension of the idea of modified biases is explaining the users' tendency to give ratings for movies they like (and rate high). We can deduce, that the reverse relationship will be analogous, and the lower is the rating predicted by our algorithm, the larger is the additional negative bias – modelling that negative bias should not largely influence recommendations. Paradoxically, the artificial example given in table 1 in [Ste10b] is close to MAR (missing at random – $p(Selected|Data) = p(Selected|Data_{observed})$), illustrating that in some cases a lot about the missing data structure can be deduced from the observed ratings. It would be difficult to prove that real-life data in a collaborative filtering setting is NMAR (not missing at random), because there is always a possibility left that an undiscovered relationship between the observed and the missing data will explain completely the probability of missing data, rendering the unobserved data useless). It is reasonable to suspect though that, in the Netflix dataset, part of the structure of missing data is unexplainable by the observed data.

The evaluation measure suggested by Netflix, RMSE on the set of most recent ratings, would probably do a good job evaluating a deployed recommender system, balancing between evaluating precision (by measuring accuracy for movies proposed by the recommender system) and recall (by measuring accuracy for movies found by the user, for example, through search). After deployment, the training set changes with new ratings added over time, and the distribution of ratings in the whole set (together with the distribution of which items are selected to rate) over time becomes similar to the set of most recent ratings. For a recommender system before deployment, RMSE on the most recent ratings from the database does not necessarily give accurate answers for questions, whether the system does not recommend dubious items (does it have high precision?) or whether it identifies the relevant items among all items (does it have high recall?). Other commonly used evaluation measures (including the ranking-based) calculated on gathered user-selected ratings have similar disadvantages. The evaluated prediction algorithm can produce any number of false positives on the missing data, and if we measure error on observed data, without accounting for the missing data structure, the real error can be unrelated and arbitrarily high. Attempts to define a measure evaluating personalized rankings of items on data distribution are not satisfying so far, like for example, the algorithms CoFiRank [Wei07] (does not take into account the missing data structure, and the optimization is complicated), Bayesian Personalized Ranking [Ren09], or the task Track 2 of the KDD Cup 2011 [Dro11, McK11, Lai11, Jah11b, Mni11], which evaluates how well the algorithm distinguishes high ratings from missing data sampled according to the frequency of items (an evaluation criterion rewarding both predicting high ratings, and predicting the missing data structure). Skipping information contained in low and average ratings is controversial, as well as highly rewarding accurate prediction of the missing data structure, which can depend on the mechanism of website navigation,

website layout, previously given recommendations, and other causes that not necessarily indicate user preferences (for example, currently the search option on YouTube weights highly the number of views, instead of giving more weight to likes and dislikes, and, in effect, videos with a misleading title, tags, and thumbnail because of traffic from search can have millions of views). Another approach [Ste10b] is treating missing data as additional ratings ≈ 2, weighted about 20 times less in regression than the regular, observed ratings. – the observed improvement of ATOP (area under the top-k recall curve) accuracy of this method should be attributed to penalizing unpopular items, and hence rather attributed to accounting for variance of predictions than to accouting for missing data.

Said all that, because I do not have additional data, in this work I stay with evaluating methods on the Netflix Prize criterion of RMSE on the test set, hence obtaining methods that predict the expected rating on the observed data distribution. Predictions of these methods, to produce accurate recommendations, need to be adjusted to account for uncertainty of predictions and for the missing data structure. The adjustments are heuristic, and likely need to be tuned on additional data. Additional data can be also used to develop algorithms with domain adaptation, that optimize the proper evaluation criterion (which needs yet to be identified) directly. I anticipate that such algorithms with domain adaptation will be close to the algorithms developed in this work.

For different choices of the hold-out set, RMSE will differ. I will list all RMSE variants appearing in this work, and also RMSE variants that often appeared in other works dedicated to the Netflix Prize task:

- RMSE_{qual} - one half of qualifying.txt (which has the same distribution as probe.txt) is used as the hold-out set. This evaluation score, called also the quiz score, was reported by the automatic system evaluating the Netflix Prize submissions, and is the most common kind of RMSE score appearing in papers describing solutions for the Netflix Prize task. We should emphasize, that including probe.txt in the training set largely improves the RMSE_{qual} score.
- RMSE_{test} - another half of qualifying.txt set (the rest of the $2,817,131$ ratings after excluding the quiz set). RMSE_{test} was used as the final score in the Netflix Prize competition. RMSE_{qual} was lower from RMSE_{test} by 0.0006-0.0014, the largest difference being for those of top teams, who combined their solutions using quiz scores (RMSE_{qual}) for individual methods, this way causing overfitting.
- RMSE_{probe} - probe.txt, having $1,408,395$ ratings, is used as the hold-out set. The probe and the qualifying set come from the same distribution and contain max. 9 latest ratings of each user (randomly split in 1:2 ratio between the probe and qualifying set, and the qualifying set being further split to the quiz and test set). Some teams used probe.txt as a hold-out set for initial training of methods and for computing blending parameters, with further retraining each method on the whole training.txt set, and combining the resulting predictions using the blending parameters calculated earlier on the probe set. There are differences in distribution of probe.txt (and qualifying.txt), and the rest of training.txt without probe.txt . The most visible differences are in the distribution of user support (see section 3.5 "A closer look at the data"), the distribution of rating dates, and the difference between the averages of the training and probe set (section 4.2.1), but, with minor exceptions, no way was found to use these differences to improve accuracy of more complex methods.
- RMSE_{15} - is the RMSE score most frequently appearing in this work. Using the whole probe.txt as a hold-out set was an inefficient use of data. A better idea was to include most of probe.txt in the training set. The chosen proportion 15% (211365 ratings) was a balance between using the largest possible hold-out set to blend many methods, and using the largest possible training set, to obtain good predictive accu-

racy. The same RMSE_{15} score was used in the earlier paper [Pat07]. An advantage of using RMSE_{15} is that methods are trained only once, while not losing too much accuracy for removing 15% of the probe set for training (it was much better than removing 100%). RMSE_{15} of individual methods was lower by $0.0025 - 0.0035$ than RMSE_{qual}. The difference was larger for the blending, because it introduced additional overfitting.

- RMSE_{10} - a convenient idea of Gravity team was such drawing the 10% of probe.txt, so that $\text{RMSE}_{10} \approx \text{RMSE}_{qual}$ [Tak08a].
- RMSE_{train} - one can speak also about measuring RMSE on training set, but this value does not mean much, because of the occurring overfitting the data. Because the degree of fitting the data varies between the methods (especially apparent in kernel methods, which fitted the data very closely), RMSE_{train} cannot be used to compare methods or to tune parameters of a method. In this work training RMSE will not be reported.
- RMSE_{small} - was calculated on a subset of the Netflix data, described in the previous section 3.3. The subset contains $10,059$ users and 500 movies, and is 56% dense. Of it, $10059 * 500/2$ ratings are used as the training set, and the remaining ratings become the test set, for which the RMSE is computed.

We should note that a RMSE value is data specific, and of course there is no point to compare RMSE computed on different datasets, or to give RMSE of a single method on a rare or unpublished dataset (this value will not tell much to the reader about the accuracy). RMSE computed on a fixed hold-out set serves only to compare different methods on the same dataset. One advantage of focusing on prediction for the Netflix Prize task is, because of its popularity, the possibility to compare results with many methods developed by others.

In general, instead of using one hold-out set, it is better to use cross-validation, that is, to repeat learning for several hold-out sets, and average the results. Because of the size of the Netflix Prize dataset, most methods took long time (5-15h) to learn, and it was more convenient to run the methods only once on a chosen single hold-out set.

The Bellkor team noticed, that it is possible to blend the methods learned on all available data, without an additional held-out test set [Tos09]. The trick is to use RMSE_{qual} for each method, returned by the automatic Netflix Prize evaluation system. The remaining statistics used to calculate the linear blend are possible to compute on the training set.

Imprecisions in the choice of task (and the awareness, that the task is not 100% relevant for the goal of building a good recommender system) are not a large obstacle, because the main goal of this work is developing and systematizing general principles, insights, methodologies for solving a real-life prediction task. The task of minimizing RMSE on a hold-out set is, in a way, elegant because of its minimalism and simplicity. It is "clean" for a task concerning real-life data. Throughout the work we will focus on the aspect of prediction of held-out data, and we will assume, that RMSE on held-out data is a choice good enough as a criterion of evaluating accuracy. If (unlikely) it would turn out, that the modelling the missing data structure is of large importance for quality of recommendations, it does not matter much for conclusions in this work, because the center of our interest is closer to searching for most accurate prediction methods for a fixed task (to better understand the prediction domain, generalize, develop general approaches for different prediction tasks), than obtaining best possible usability in one fixed real-world application.

Summarizing shortly, we will minimize RMSE for held-out observed data. This criterion will allow us to sufficiently explain the structure of data. To use the developed methods in a recommender system it is necessary to correct the expected predicted rating by penalizing large variance of prediction (for example, by subtracting the multiplicity of

estimated standard deviation of prediction, see sections 4.2.4, 6.1, 6.2.3). In my experience, using an accurate method optimizing RMSE, like the regularized SVD, with correction for variance of predictions is good enough to create a reasonably accurate recommender system. Additionally, penalizing the likely missing data (for example with the adjustments described in section 6.2.3) can possibly improve recommendations.

3.5 A closer look at the data

The Netflix Prize data contains $100,480,507$ ratings on a scale 1-5: $17,770$ movies rated by $480,189$ users. I will look at the four variables contained in the data: ratings, users, movies and dates. Through visualization of the first observed distributions in data I will try to give the reader a perspective on what the Netflix Prize data contain, before going on to the description of algorithms that use more complex dependencies and effects in data. Most of the plots and tables in this section appeared earlier in a similar form in other articles.

The distribution of the training set (training.txt without probe.txt) differs from the distribution of the probe and qualifying sets (probe.txt and qualifying.txt), and I will sometimes do separate visualizations or summaries for the training and probe sets. The main difference between them is the distribution of the number of user ratings. In the training set users have varied number of ratings, but in the probe and qualifying sets ratings are chosen uniformly: most users have exactly 9 ratings total in both sets. Distributions of rating dates also differ – the probe and qualifying sets contain the most recent ratings of each user.

The training set will be denoted in short as Tr, the probe set as Pr, and the qualifying set as Qu. The training set (Tr) contains $100,480,507 - 1,408,395 = 99,072,112$ quadruples (rating, user, movie, time). The probe set (Pr) contains $1,408,395$ quadruples (rating, user, movie, time). The qualifying set (Qu) contains $2,817,131$ triples (user, movie, time). The ratings for qualifying set were disclosed after the end of the Netflix Prize contest, but here I treat those ratings as unknown.

Figure 3 shows the proportions of each rating $1-5$ in the dataset. The distributions in the training set and in the probe set are visibly different.

Figure 3: Rating proportions in the training set and in the probe/qual set

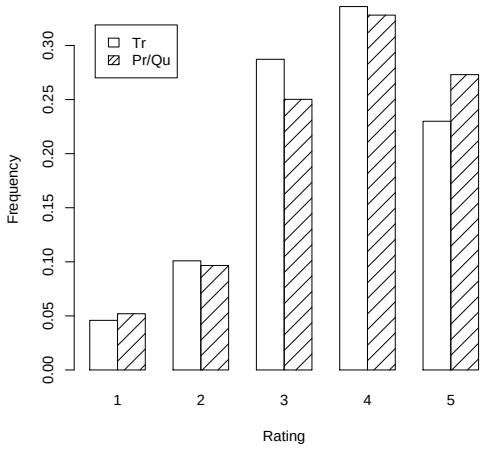
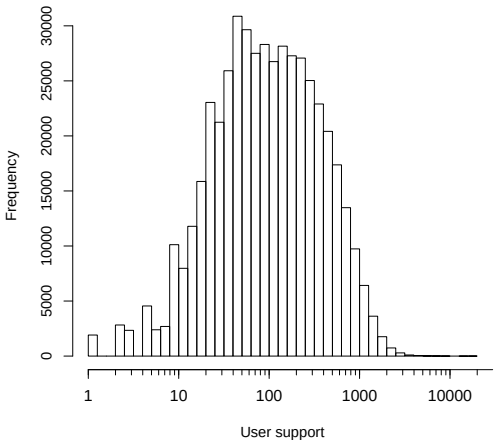


Figure 4: Histogram of log(user support) in the training set



The distribution of gathered user preference data can differ considerably between different datasets. First, different types of user preferences can be gathered. Most popular are ratings, possibly with an additional option "not interested". Other popular interfaces

are "like" buttons, or a binary like vs. don't like choice. Ratings may be collected on different scales: $1-5$ or $1-10$, on an integer scale, or with 0.5 steps. If ratings are collected on the same scale, the distribution can differ depending on the type of items, the way of data collection, the user interface, the choice of items presented to a user for rating, or depending on whether are they ratings from a recommender system, ratings from voting for top items, or ratings given to help in determining a collective opinion about an item.

The article [Mar09] compares distributions of ratings in several datasets: the Yahoo! dataset of music ratings, and the EachMovie, MovieLens and Netflix datasets containing movie ratings. The Yahoo! dataset contained both user-selected items, and items selected randomly. In the Yahoo! dataset in the user-selected part most frequent were ratings 1 and 5, on a $1-5$ scale, but for randomly selected items the distribution of ratings was very different: more than 50% of ratings were ones, and less than 5% of ratings were fives. Clearly, it is a large difference from a scenario of data missing completely at random – there is a large bias towards selecting for rating items liked by the user (and highly rated). In a real recommender system predictions are usually made for all items, so predictions for a missing completely at random (MCAR) setting seem more appropriate than predictions for user-selected data (on the other hand, it may be possible to sufficiently explain the missing data mechanism on the basis of the observed data, because users tend to rate movies they like). The Netflix Prize data provides only user-selected test data, so we cannot observe distributions and evaluate the trained methods on data randomly missing, but the phenomenon of selection bias can be observed indirectly, as seen later in figure 15, that shows decreasing average user rating with increasing number of movies rated. The conclusion from figure 15 is that if a user were forced to rate all movies, his average rating would be very low. It requires further analysis to say if recommendations in a real recommender system should be corrected by decreasing the predicted rating for unlikely seen items. Such a correction was used in a recommender system described in section 6.2.3.

Let's look at the distribution of the number of ratings of each user. In the training set there are few users who rated many movies and there are many users who rated few movies. The histogram of logarithm of user support in figure 4 resembles a normal distribution. Figure 5 shows the number of users with given support in combined training, probe and qualifying sets. Figure 6 is a zoomed version of figure 5 for users with support < 100.

Figure 5: Count of users with given support (support < 1000)

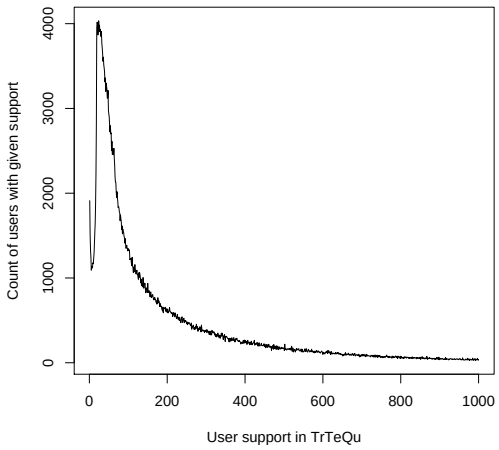


Figure 6: Count of users with given support (support < 100)

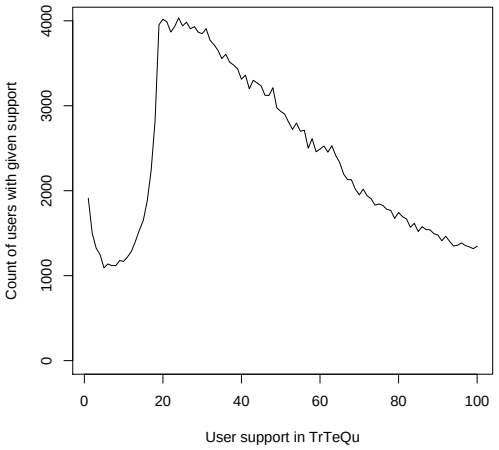


Table 1 shows the number of ratings in the set Pr+Qu. Most users have exactly 9 ratings in the Pr+Qu set, so we can say that, in our task of minimizing RMSE on the test set, the accuracy of rating prediction of each user is equally important. In contrast, for

Table 1: User support in Pr+Qu

1	2	3	4	5	6	7	8	9	Sum
3411	2572	2230	2237	2346	2509	2940	3527	458417	480109

the calculated on the training set RMSE_{train}, users who rated more movies have a larger share in the mean squared error.

Now let's look at the distribution of the number of ratings of each of $17,770$ movies. We can distinguish popular, mainstream movies and movies from the "long tail". Table 2 shows the most popular 10 movies. Figure 7 is a histogram of the logarithm of movie support in the training set, and figure 8 is the same histogram for the Pr+Qu set. Unlike for users, for movies the two sets have similar a distribution. In the task of minimizing RMSE on the qualifying set more important is accurate prediction for a popular movie than accurate prediction for a rarely rated movie.

Table 2: Most popular movies in the dataset

Rank	Title	Average rating	Count
1.	Miss Congeniality	3.359	227715
2.	Independence Day	3.724	216233
3.	The Patriot	3.783	200490
4.	The Day After Tomorrow	3.443	194695
5.	Pirates of the Caribbean: The Curse of the Black Pearl	4.153	188849
6.	Pretty Woman	3.901	190320
7.	Forrest Gump	4.299	180736
8.	The Green Mile	4.307	180883
9.	Con Air	3.454	177825
10.	Twister	3.412	177212
	average of top 10	3.783	193495.8
	average overall	3.603	209.3

Figure 7: Histogram of log(movie support) in the training set

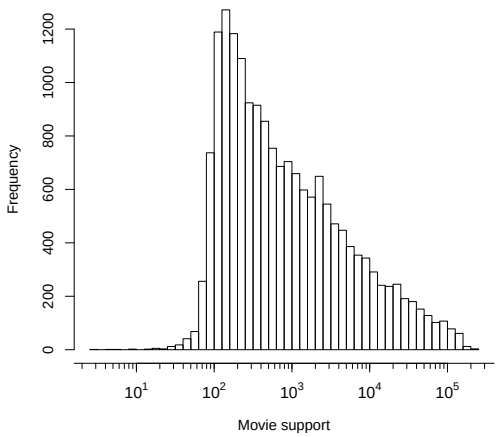
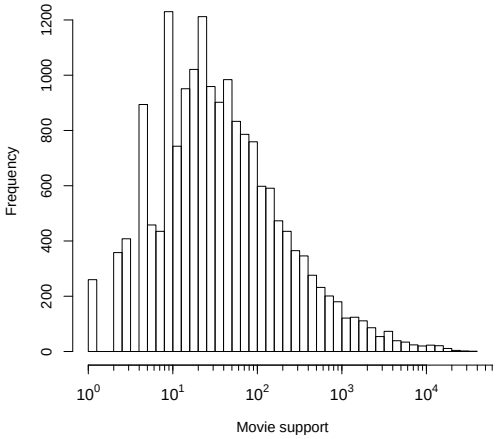


Figure 8: Histogram of log(movie support) in the probe/qual set



The observed shape in figures 7 and 8 suggests that movie support follows a power law distribution, but the log-log plot (figure 9) shows a relationship different from linear

between the logarithm of the rank of movie support and the logarithm of movie support. In figure 10 a relationship close to linear was obtained by plotting the square root of movie rank against the logarithm of movie support. Hence the law that explains the behavior of the long tail in the Netflix Prize data is described by a distribution with pdf approximately $p(n) \propto e^{-a\sqrt{n}}$, where n is the movie frequency rank, and $p(n)$ is the movie frequency in the dataset, so a different distribution than the commonly observed power law distribution $p(n) \propto n^{-a}$. It should be noted here, that Netflix added new movies to their offer gradually. The Netflix collection contained about 17770 DVD's in 2005 year, available to rent and rate, comparing to 4470 movies in 2000 year [Tan09] (we can speculate, that it was not a uniformly random subset of the 17770 movies, but biased towards selecting more popular movies – popular movies were available for a longer time, further amplifying their popularity).

Figure 9: log(movie rank) vs. log(movie support)

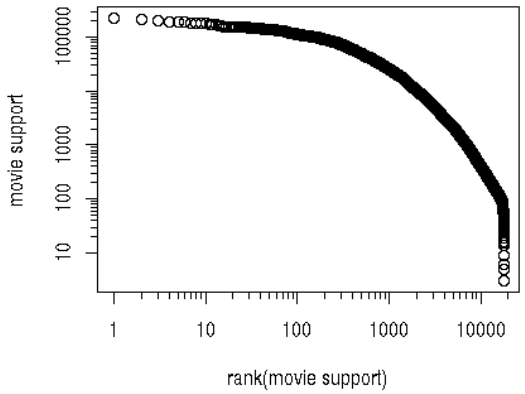


Figure 10: sqrt(movie rank) vs. log(movie support)

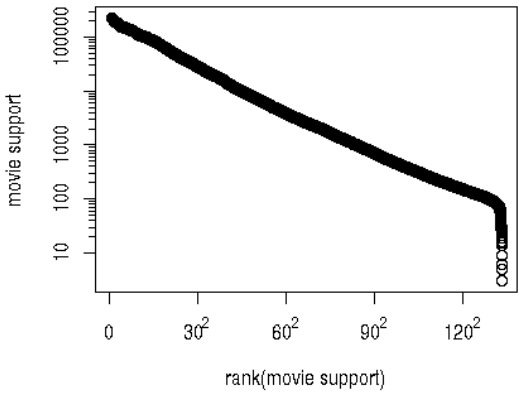


Table 3 shows the distribution of users and movies in the training set. Users were split according to their support (the number of ratings) into 10 equinumerous groups. Similarly, movies were split into 10 groups according to their support.

Table 3: Distribution of users and movies in the training set.

	1	2	3	4	5	6	7	8	9	10	Sum
1	30.54%	7.46%	2.92%	1.39%	0.71%	0.41%	0.25%	0.17%	0.12%	0.09%	44.04%
2	16.32%	2.13%	0.75%	0.37%	0.20%	0.11%	0.07%	0.05%	0.03%	0.02%	20.05%
3	10.37%	1.13%	0.44%	0.24%	0.14%	0.08%	0.05%	0.04%	0.03%	0.02%	12.53%
4	6.89%	0.69%	0.29%	0.16%	0.10%	0.06%	0.04%	0.03%	0.02%	0.01%	8.29%
5	4.60%	0.45%	0.20%	0.12%	0.07%	0.04%	0.03%	0.02%	0.01%	0.01%	5.56%
6	3.09%	0.30%	0.14%	0.08%	0.05%	0.030%	0.020%	0.014%	0.010%	0.006%	3.74%
7	2.13%	0.19%	0.09%	0.05%	0.03%	0.019%	0.013%	0.009%	0.006%	0.004%	2.55%
8	1.46%	0.12%	0.06%	0.03%	0.02%	0.012%	0.009%	0.006%	0.004%	0.003%	1.73%
9	0.89%	0.08%	0.04%	0.02%	0.01%	0.009%	0.006%	0.004%	0.003%	0.002%	1.07%
10	0.36%	0.04%	0.02%	0.01%	0.01%	0.005%	0.003%	0.002%	0.002%	0.001%	0.45%
Sum	76.66%	12.59%	4.93%	2.47%	1.33%	0.78%	0.49%	0.34%	0.24%	0.16%	100.00%

The already mentioned long tail situation is visible: in the training set the 10% most frequent users gave 44.04% of all ratings, and the 10% most frequent movies received 76.66% of all ratings.

Table 4 shows an analogous distribution of users and movies in the Pr+Qu set. Users and movies were split into groups according to their support in the training set.

Next plots explore the date coordinate. Figure 11 shows count of ratings on each day

in the training set. Figure 12 is the same as the previous one, but for the Pr+Qu set. Figure 13 shows a histogram of the logarithm of the number of ratings on each day, in the training set.

Figure 14 shows the number of ratings on each weekday. Most ratings were given on Tuesdays, and least ratings on Saturdays and Sundays. This contrasts with traffic statistics of the Netflix.com website, which show that the highest traffic on the website is on Saturdays and Sundays (this can be explained by the fact that in the Netflix system, after a user returns a DVD, he gets an e-mail encouraging to rate the movie).

Table 4: Distribution of users and movies in the probe and qualifying sets.

	1	2	3	4	5	6	7	8	9	10	Sum
1	5.87%	2.13%	0.87%	0.53%	0.34%	0.18%	0.14%	0.09%	0.06%	0.06%	10.26%
2	6.89%	1.69%	0.66%	0.37%	0.23%	0.14%	0.10%	0.06%	0.05%	0.05%	10.24%
3	7.26%	1.47%	0.58%	0.35%	0.22%	0.13%	0.09%	0.07%	0.05%	0.05%	10.27%
4	7.47%	1.32%	0.56%	0.32%	0.21%	0.13%	0.10%	0.07%	0.05%	0.05%	10.27%
5	7.59%	1.24%	0.53%	0.33%	0.21%	0.13%	0.10%	0.07%	0.05%	0.05%	10.30%
6	7.63%	1.21%	0.54%	0.33%	0.21%	0.13%	0.09%	0.06%	0.05%	0.05%	10.30%
7	7.90%	1.11%	0.50%	0.30%	0.19%	0.12%	0.09%	0.06%	0.04%	0.04%	10.35%
8	8.15%	0.99%	0.45%	0.26%	0.17%	0.11%	0.07%	0.05%	0.04%	0.03%	10.34%
9	8.21%	0.92%	0.42%	0.26%	0.18%	0.11%	0.07%	0.05%	0.04%	0.03%	10.30%
10	5.57%	0.75%	0.38%	0.23%	0.15%	0.10%	0.07%	0.05%	0.04%	0.03%	7.38%
Sum	72.54%	12.84%	5.47%	3.30%	2.10%	1.28%	0.92%	0.64%	0.48%	0.43%	100.000%

Figure 11: Count of ratings in time – training set

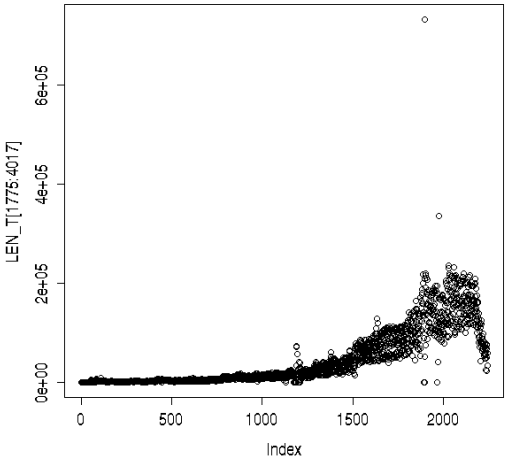


Figure 12: Count of ratings in time – probe and qualifying set

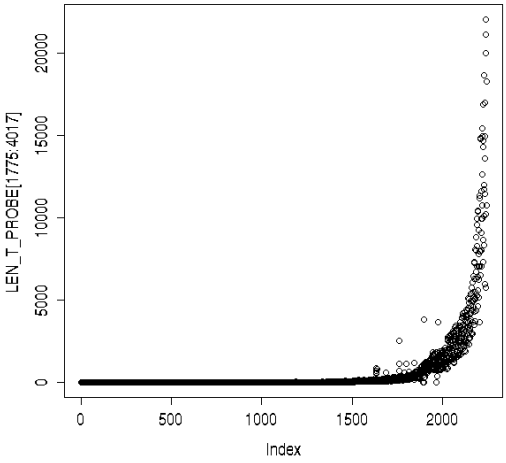


Table 5 shows the percentage of ratings given each year in the training set, for each of 10 equinumerous groups of users, split according to the decreasing number of ratings of a user. Ratings from 2005 dominate the dataset, taking between 43% and 72% of all ratings in different groups of users. Table 6 is the same, but for Pr+Qu data.

Because this paper addresses mainly rating prediction, we will look next how ratings change along with the remaining three variables: users, movies and date.

Figure 15 shows the average rating changing with the user support. We see that the average decreases with increasing number of ratings given. A fitted linear relationship is marked on the plot (note that on the right side of the plot it may be inaccurate). Assuming that the average decreases approximately linearly with the increasing user support, we can try extrapolating the average on a situation, when a user rated all $17,770$ movies. In this case it seems that the mean user rating would be less than 2 on average. A similar pattern with the average user rating decreasing with increasing number of ratings was also

Figure 13: Histogram of log(ratings per day) in the training set

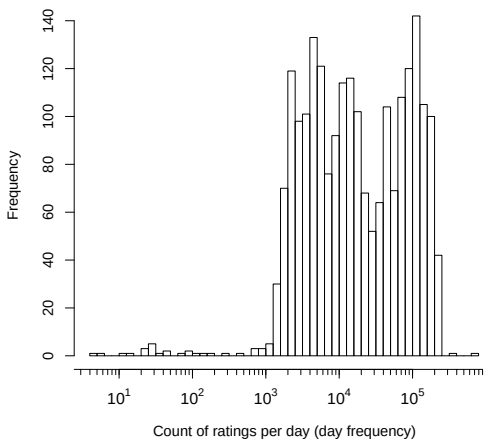


Figure 14: Ratings count per weekday

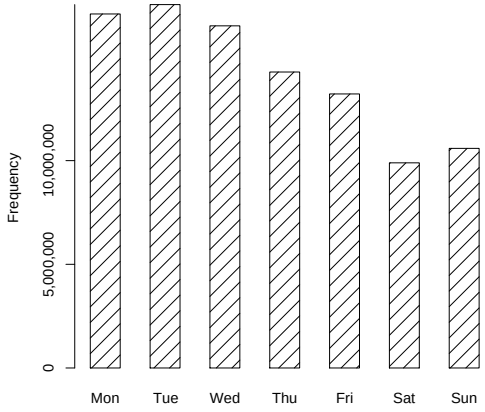


Table 5: Distribution of the rating year in the training set, for users grouped by support (10 groups)

	1999	2000	2001	2002	2003	2004	2005	Sum
1	0.003%	1.333%	2.377%	5.862%	12.298%	34.444%	43.683%	100.0%
2	0.002%	0.788%	1.669%	4.003%	9.205%	30.006%	54.326%	100.0%
3	0.001%	0.633%	1.337%	3.323%	8.702%	27.940%	58.064%	100.0%
4	0.000%	0.538%	1.228%	2.879%	8.172%	26.878%	60.305%	100.0%
5	0.001%	0.480%	1.046%	2.597%	7.718%	25.934%	62.223%	100.0%
6	0.002%	0.425%	0.909%	2.321%	7.049%	23.379%	65.915%	100.0%
7	0.002%	0.371%	0.726%	1.931%	5.885%	20.239%	70.847%	100.0%
8	0.002%	0.367%	0.670%	1.766%	5.793%	19.471%	71.930%	100.0%
9	0.001%	0.347%	0.737%	1.910%	6.278%	20.983%	69.745%	100.0%
10	0.000%	0.308%	0.762%	2.059%	6.206%	20.776%	69.888%	100.0%

Table 6: Distribution of the rating year in the probe and qualifying sets, for users grouped by support in the training set.

	1999	2000	2001	2002	2003	2004	2005	Sum
1	0.002%	0.954%	1.833%	4.271%	9.642%	30.152%	53.146%	100.0%
1	0.003%	1.079%	1.882%	5.130%	11.491%	31.525%	48.889%	100.0%
1	0.002%	0.720%	1.447%	4.615%	12.018%	31.762%	49.436%	100.0%
1	0.002%	0.456%	1.118%	3.630%	10.565%	31.744%	52.486%	100.0%
1	0.002%	0.405%	1.094%	3.317%	10.177%	29.965%	55.040%	100.0%
1	0.003%	0.535%	1.245%	3.926%	11.557%	29.816%	52.919%	100.0%
1	0.003%	0.661%	1.406%	4.837%	12.211%	28.308%	52.574%	100.0%
1	0.003%	0.644%	1.455%	5.078%	11.629%	28.331%	52.861%	100.0%
1	0.003%	0.720%	1.491%	5.422%	11.804%	26.706%	53.854%	100.0%
1	0.008%	1.232%	2.241%	8.314%	13.252%	20.787%	54.166%	100.0%

observed in [Dro11] in a dataset with music ratings. The drop of average can be explained by a strong tendency of users to rate items they like. If a user rates more movies, then he is forced to evaluate movies he does not like and his rating average decreases. The article [Mar09] describes the case of Yahoo! Music data, where with uniformly random selection

of an item for rating, the most frequent rating was the minimal rating 1.

Figure 15: User support vs. user mean rating

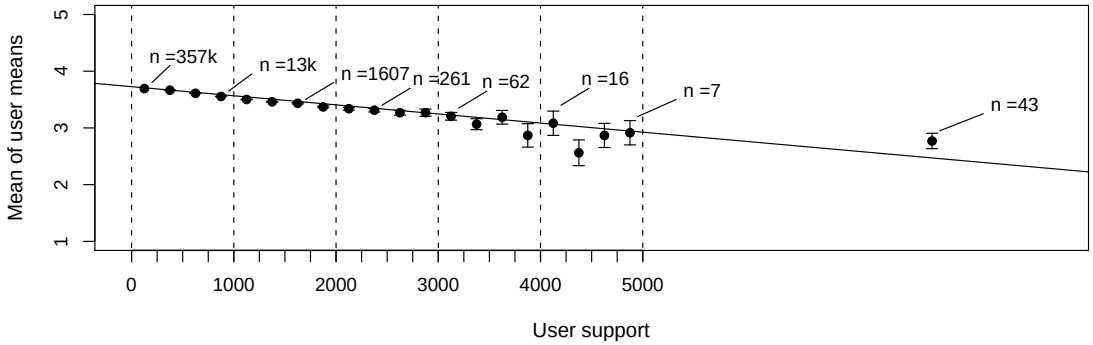


Figure 16 shows, that the average rating increases with increasing movie support. The relationship between the average rating and the logarithm of movie support is approximately linear.

Figure 16: Log movie support vs. movie mean rating

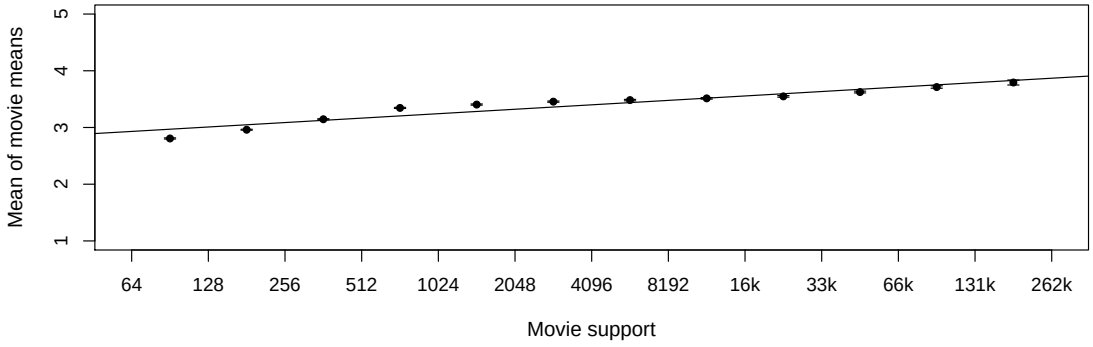
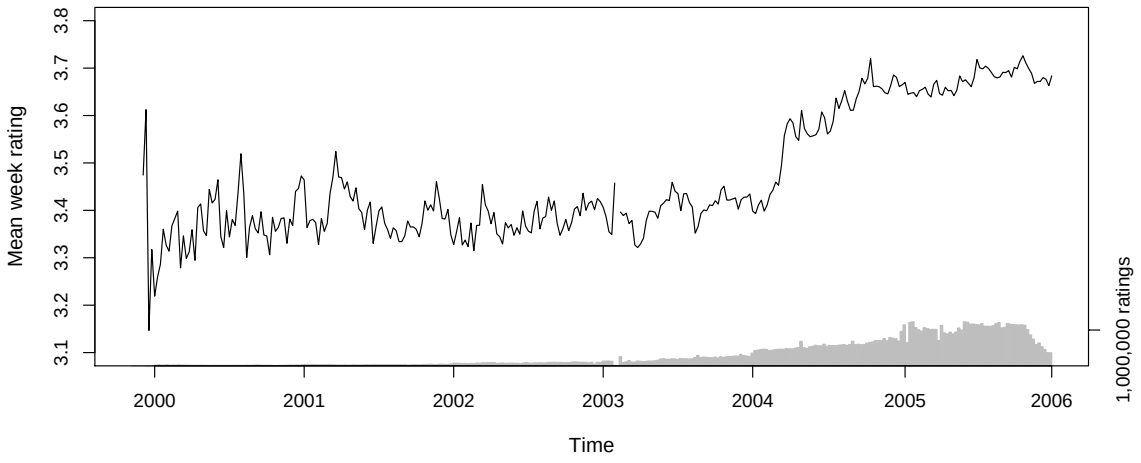


Figure 17 shows the average rating in time. We notice a jump of average rating in early 2004, which becomes explained by modelling the movie mean. A probable explanation for the jump is adding highly rated TV series and movies to Netflix's selection in early 2004 (I have not verified this presumption). Another possible explanation are changes in the Netflix's rating interface (also, not verified).

Figure 17: Mean rating in time



I have listed the first, basic visualizations and summaries of the dataset. As it will be seen in later sections, only some of these plots are relevant for the task of maximizing predictive accuracy, that we are primarily interested in. Nevertheless, presenting and commenting the basic effects, easiest observed in data consisting of just four variables: ratings, users, movies, and date, can be helpful.

The simply put task of maximizing predictive accuracy by modelling $E(rating|user, movie, time)$ or $E(rating|user, movie)$ is deeper. As we shall see later in this work, most important for accuracy is modelling a large number of hidden variables, capturing such properties of data as similarity between items and similarity between users, mainly through dimensionality reduction. Different ways of explaining hidden structure of matrix or tensor-type data of this type constitute the field called collaborative filtering.

"Free your mind Luke Skywalker"

vzn

4 Methods of rating prediction

Now I proceed to the main part of the work, a review of collaborative filtering methods predicting movie ratings. As explained in section 3.4 "Evaluation", we are interested in methods with the best predictive accuracy, as measured by RMSE (root mean squared error) on a certain subset of the Netflix Prize data, held out during training and not used to learn the parameters of the method.

First, I analyze the simplest methods: global mean only, models with movie biases and user biases, regularized SVD with one feature. Restricting the scope to the simple models allows to stay close to probabilistic modelling and the Bayesian approach, and allows to pay attention in which places I move away from that methodology. Fundamental questions will appear, which need to be answered for any models realizing the same task. Are differences between training and tests sets significant from the point of view of prediction accuracy in our task? Should I take into account in the model the missing data structure? Was the model identified correctly? Do I use the right prior distribution for parameters? Do I model the output (response) variable well? Is the structure of hidden variables in the model chosen right? Is the multiplication in the SVD model the right way to combine hidden variables? I will compare different ways of realizing parameter inference for the simple models: is it best to approximate the Bayesian approach with MCMC with Gibbs sampling, or with Variational Bayes, or with neural networks-like methods? What is the best way to regularize the parameters in the NN approach? I will reflect on a question going beyond the chosen dataset and prediction task: how to adapt an algorithm minimizing RMSE to calculate lists of recommendations?

After looking at the simple models I will go on to more complex methods, that give the best predictive accuracy among the known methods. The choice of methods and combinations of methods to implement and include in my ensemble was based mostly on:

- experimental experience,
- understanding of the structure of the data and the observed patterns,
- guidance by known situations from machine learning and probability,
- systematic search of variants of methods to minimize the test error of a method, and the test error of the ensemble (mostly search "by hand", but sometimes automatic).

Described will be mainly (with a few exeptions) methods that improved the accuracy of the ensemble – and these were fewer than 10% of all implemented variants of methods. While developing the methods I did not have in mind the didactic perspective, but only the ensemble performance.

First, the multidimensional regularized SVD model (called also a matrix factorization) will be thoroughly described. In the regularized SVD the hidden per-movie variables represent automatically learned analogues of movie genres, and the hidden per-user variables state the user's preferences for the automatically indicated genres. SVD results can be postprocessed by other methods to improve accuracy, and also SVD features have applications beyond prediction: to find similar movies, similar users, cluster movies or users, visualize, suggest new genres, etc. (see section 4.3.4 and chapter 6). I will discuss exploiting the missing data structure, and using time effects to improve accuracy. I will briefly summarize the matrix norm regularization form of the regularized SVD.

Next I move on to a short description of nonlinear methods, among which most efficient were Restricted Boltzmann Machines. Described will be distance-based methods, such as K-nearest neighbors and kernel methods like Gaussian Processes [Ras03, Ras06], and a few other methods, for example, K-means.

There were attempts to augment the Netflix Prize dataset with external item metadata, but it turned out, that ratings in the Netflix dataset carry a sufficiently large amount of information, so that additional item metadata did not improve prediction. I will write about my attempts of using the IMDb and Wikipedia metadata.

Finally, I will review different ways of combining prediction methods: preprocessing and postprocessing one method with another, integrating methods, and blending of separately learned methods. The final solution with accuracy 8.48% better than the baseline Netflix algorithm was a blend of 69 methods. In practice of building a recommender system, one method should suffice to obtain satisfying accuracy, and, from the point of view of software engineering and quality assurance, it is best to use only one selected prediction method.

To be easier to reproduce the methods and verify the accuracy, in the first part "Simple models" 4.2 the accuracy will be measured by RMSE_{probe} (the entire probe set will serve as a hold-out set). In the remaining parts of the chapter RMSE_{15} is used for evaluation (see section 3.4). The main advantage of using RMSE_{15} was that it allowed to train methods only once, at the cost of small decrease of accuracy in comparison with re-training methods on the entire training set.

The methods listed were developed to predict movie ratings, but they seem to work equally well when applied to predicting ratings for any type of items – see, for example, the experiments with predicting music ratings [Che11a, Jah11a, Jah11b].

4.1 Notation

The following notational conventions are used throughout the work:

R - the sparse matrix of ratings $1-5$
S - the matrix of binary indicators, which movies were rated by which users
r_{ij} - the rating given by user i for movie j
$\hat{r}_{ij}$ - the predicted rating
$y_{ij} = r_{ij} - \hat{r}_{ij}$ - residuals of a predictor
$\hat{y}_{ij}$ - predictor of a residual (usually a rating with global effects subtracted)
$N = 408189$ - the number of users
$M = 17770$ - the number of movies
J_i - the set of movies rated by user i
I_j - the set of users who rated movie j
$M_i = |J_i|$ - the number of ratings given by user i
$N_j = |I_j|$ - the number of ratings for movie j
J_{it} - the set of movies rated by user i on day t
$I_{j_2 j}$ - the set of users who rated both movie j and j_2
c_i - the bias of user i
d_j - the bias of movie j
u_{ik} - the preference of user i for feature k
v_{jk} - the value of feature k of movie j
$\mathbf{u}_i$ - the vector of preferences of user i
$\mathbf{v}_j$ - the vector of features of movie j
i - index for users
j, j_2 - indices for movies
k - index for hidden features

Vectors are always vertical $n \times 1$, like in the book [Has09] or in the R language environment. For example, when we have a matrix X of dimensionality $n \times p$, the vector $\mathbf{x}_i$

denoting the i-th row, is vertical $p \times 1$, not horizontal $1 \times p$. Vectors are denoted by bold letters.

Real values are rounded to between one and six digits, most often to four digits.

The word "empirical" denotes estimates observed in the training set, e.g. empirical mean, empirical probabilities.

The word "support" denotes frequency, count of data in the dataset, e.g. user support is the number of ratings given by the user.

4.2 Simple models

I begin presenting the rating prediction methods with simple models: the models of biases and a highly simplified regularized SVD model. Constraining the number of parameters limits possible model structures, making it easier to explore the space of possible models, parameter distributions, interactions between parameters. In the simple models some essential concepts in rating prediction are visible more clearly. Conclusions from the analysis of the simple models should to a large extent hold for more complex models, since emerging questions and decisions to make in the complex models are similar. I am interested here primarily in analysing the regularized SVD algorithm, explaining why this model works so well, and deciding whether it is close to the unknown optimal model for the data, or maybe it is possible to improve it.

Section 4.2.1 starts with approximating the entire dataset with one variable, with the goal of minimizing generalization RMSE. It turns out, that even in this simplest model there are a lot of questions and issues to consider, which will appear also later in complex models. I will ponder, which type of point estimation to choose. Is it better to use Bayesian or classical statistics? How to choose a prior? What to do with the visible differences between the training and test sets? Are the differences significant from the point of view of prediction? Is it good to treat the test set, as if it came from the same distribution as the training set, or maybe prediction accuracy can be improved with domain adaptation methods?

Section 4.2.2 discusses output modelling. The ratings in the Netflix Prize data are integers on the scale $1-5$. Most often the output (predicted rating) was approximated as a clipped normal distribution. I will analyze if this is the best approach, whether some important patterns in data are not skipped, which proper modelling could improve prediction accuracy. Alternatives are modelling the output as a single variable from binomial distribution, or modelling each rating separately. To get further insight hierarchical probabilistic models were trained with nonparametric priors, and the empirical distributions in the generated data were compared with the empirical distributions in the Netflix dataset.

In section 4.2.3 I discuss a simple model of two biases: user bias and movie bias. The discrete output $1-5$ will be approximated by a normal distribution. Different ways of learning will be considered, either treating biases as single values or, in the Bayesian approach, as random variables. Hierarchical probabilistic modelling will be used, with joint priors for all users and all movies. Learning models by alternating point estimation will be compared to methods keeping track of the whole distributions. Assumptions of the model will be questioned, such as exchangeability of users and of movies, using Gaussian distributions as priors for the hidden parameters, or combining the hidden parameters with multiplication. I will compare posterior distributions appearing in a parametric approach with Gaussian priors with a nonparametric approach.

The discussion on the model of biases is an occasion to reflect how to prepare a global list of top items, a task addressed in the following section 4.2.4 "Which movie is the best?". The task of creating a global list of recommendations is relevant for us, because it is similar to generating personalized lists of recommendations. Considerations about how to adapt prediction algorithms that minimize RMSE to calculate recommendations are a

natural complement connecting the prediction methods described in the work with the context of applications in recommender systems. To properly position an item on a list, uncertainty of prediction should be taken into account. We will think also on the influence of the missing data structure.

In section 4.2.5 I proceed to the analysis of the regularized SVD with biases in its simplest version with only one feature. The method of regularized SVD and other kinds of matrix factorizations turned out to be the most efficient approaches to the Netflix Prize task, so it is useful to examine these kinds of methods more closely, to justify the form of the SVD model and the choices made while modelling and learning the parameters. I will ask myself questions: why to use a structure with multiplying the hidden variables? Which prior distributions should the hidden variables have? I will consider different possible ways of learning the parameters: different approximations of the Bayesian approach, and different ways of regularization in the neural-networks-like approach.

To evaluate the simple methods in the following sections the entire probe set will be used as a hold-out set (accuracy measured by RMSE_{probe} instead of RMSE_{15}, used in the rest of the work).

4.2.1 Global mean

The simplest possible prediction method is approximating the entire dataset with one value. It is a toy problem, but it leads to questions with non-obvious answers, and the answers become more complex in larger models.

First let's precise the task – which mean I want to estimate. Let's say we want to estimate the expected rating in the distribution of observed data for the data we have: training.txt and probe.txt sets.

We have to digress that the global mean in the distribution of observed data is influenced by the mechanism of missing data. In this work, as an alternative for making predictions for the data distribution (training set or probe set distribution), sometimes prediction for data missing completely at random (MCAR) has to be considered, because rating prediction in recommender systems is usually applied to all items. Because the Netflix dataset does not contain a sample of MCAR ratings, there is no obvious way to estimate accurately the global mean of all data, observed and missing. Based on patterns observed in the Netflix data (section 3.5), and also in a dataset containing samples of non-user-selected ratings [Mar09], we can speculate that the global mean for all missing ratings in the Netflix dataset would be less than 2 (in a hypothetical situation, if users were forced to rate all items).

We will estimate hence the global mean for the distribution of observed data. The output variable (ratings given by users) takes integer values on a scale $1-5$. The first simplification will be treating the output as coming from a Gaussian distribution. Such simplification was made in the majority of the methods described in this work. Replacing a discrete distribution, that takes five possible values, by an unbounded continuous distribution requires longer justification and will be addressed in the next section 4.2.2.

Another large simplification here will be that the ratings are independent. This assumption is obviously false – there is a lot of structure and dependencies in data, and capturing and using those dependencies is the main theme of the rest of the work.

The next, smaller simplification is that I want to minimize average MSE (mean squared error) on test data instead of average RMSE (root mean squared error), which is the evaluation criterion in the Netflix Prize. For data of this size the difference between minimization of both criteria is very small.

First I will estimate the expected rating in combined training and probe sets. For a symmetrical and unimodal distribution (and such is the posterior distribution when we make the Gaussian simplification) the expected value coincides with maximum a-posteriori

(MAP) estimate. The Bayesian approach requires specifying prior beliefs. We have some prior expectation about where the expected value lies, e.g. because we know that the ratings are in the range $1-5$, but we use here a simple flat prior, which leads to maximum likelihood (ML) point estimation. It is one of the very few places in this work, where a method from classical statistics is good enough. Usually it is visibly better to use a non-flat prior, centered around some value (usually around zero), which results in obtaining a MAP, regularized estimate. Also, often using point estimation is a too large simplification – sometimes it is necessary to use a Bayesian or approximate Bayesian approach instead.

The resulting ML estimate of the mean value μ on the data distribution is simply the average over the combined ratings from training.txt and probe.txt sets: $R = E\hat{\mu} = \frac{1}{|Tr+Pr|}\sum_{ij\in TrPr} r_{ij} = 3.6043$.

There is more to the story. The two datasets training.txt and probe.txt have a different structure of missing data, We already observed the difference between empirical probabilities of each rating $1-5$ (figure 3 in section 4.2.2).

The average rating in the training set is: $R_1 = \frac{1}{N_1}\sum_{(i,j)\in Tr} r_{ij} = 3.6033$, $N_1 = |Tr| = 99,072,112$, and the average rating in the probe set is: $R_2 = \frac{1}{N_2}\sum_{(i,j)\in Pr} r_{ij} = 3.6736$, $N_2 = |Pr| = 1,408,395$.

One might wonder whether the difference in averages is a result of a random fluctuation or a difference in distributions. To check it, a standard statistical significance test from classical statistics [Kor06] will be good enough (proper Bayesian hypothesis testing [Mad03] may be more accurate). We make the same assumptions as earlier about independence and normality, and that the two datasets come from normal distributions with equal variance, but possibly different means. We want to check the hypothesis that the means in two datasets are equal $\mu_1 = \mu_2$ with the alternative hypothesis $\mu_1 \neq \mu_2$. The sample standard deviation on training + probe sets ($Tr + Pr$) is: $S = \sqrt{\frac{1}{N_1+N_2}\sum_{(i,j)\in Tr+Pr}(r_{ij} - R)^2} = 1.0852$, where $R = 3.6043$ is the average on $Tr + Pr$. The statistic $T = (R_1 - R_2)/(S\sqrt{\frac{1}{N_1} + \frac{1}{N_2}})$ under the null hypothesis has a t-Student distribution with $N_1 + N_2 - 2$ degrees of freedom. With so many degrees of freedom, the t-Student distribution is very close to the normal distribution $N(0, 1)$. For our data we get a very large value $T = 76.33$, which tells that the null hypothesis about the means in the two datasets being equal is almost impossible. We made a few simplifying assumptions in that test, but the conclusion about significant difference of means in the datasets would be the same if the test were done more correctly, assuming a discrete distribution, and using the Bayesian approach (we should note, that the whole idea of making a test to select one model is non-Bayesian).

Knowing that the means in two datasets differ, we can wonder whether domain adaptation can be used to learn a more precise mean for the probe set, and generally, whether the algorithms trained on the training set can be adapted to give more accurate predictions on the probe set. The general experience of practitioners was that attempts of domain adaptation did not improve prediction accuracy on the Netflix Prize dataset. It is so, because the developed models more complex than the single global mean have enough structure to explain differences between the training and Pr+Qu set. The only methods used that can be seen as domain adaptation were blending on the test set the predictors learned on the training set, and also there were small improvements of accuracy by using dates to adapt methods to the test set [Bel08].

[Dau07] points out two extreme situations in domain adaptation: 1) training data is from the same distribution as test data, or 2) training data distribution is completely unrelated to test data distribution. Usually the situation with analyzed data is somewhere in between 1) and 2). The Netflix Prize dataset is very close to the case 1). For our toy task of estimating mean of the probe set distribution certainly there is some "right" way

of using the training set, which is over 60 times larger than the probe set, but I have not performed any experiments with it (for example, training data could be used to help define a prior for the mean of the test data). [Dau07] lists and compares several possible simple methods of domain adaptation: modelling the source (training) data only, modelling the target (test) data only, modelling all data from the source and target combined, the source and target combined with different weights, using methods learned on the source data as features for the target data (blending on the test set, used extensively in the Netflix Prize is a case of this), linear interpolation of a model learned on the source data and on the target data, using the source data to determine the prior distribution for the target data. Another approach [Dau06] is a full probabilistic model for source and target data, that treats both datasets symmetrically. There are also approaches weighting the training set samples by predicted probability that the observation belongs to the training or to the test set [Bic07, Bic09], but we should note, that the training-test relationship in Netflix data is not a typical situation of covariate shift, because here the data contain only outputs, without fixed predictors, like in a typical case of linear regression. Training and test sets in Netflix data differ by the amount of observed data among users, and also by the date variable, because the test set contains the most recent ratings.

To summarize, the issues met while analyzing the simplest model of global mean were: the data is not missing completely at random, we made simplifying assumptions about normality and independence of data samples, and we observed a difference between the training distribution and test distribution, which can be ignored or one can try domain adaptation. There are also methodological questions: which general modelling approach to use? Which simplifications of the Bayesian approach to make? Is calculating the entire posterior distribution necessary, or is point estimation good enough?

All of the above questions and issues will recur in a more complex form when analyzing more complex models. Now I will look closer at approximating ratings with a normal distribution, what was done in most of the later models.

4.2.2 Discussion of output modelling

In the Netflix dataset ratings are integers on a scale $1-5$. One of the choices to make while doing predictive modelling is to decide on the output representation, that is to decide what is the result of calculations, what kind of predictions the model makes. We already mentioned that we are mainly interested in assessing the expected rating (sometimes corrected by the estimated variance of predictions), but sometimes it is justified to generate ratings as an intermediate step instead of modelling the expected rating directly. In this section I look at different possible ways of output modelling for the Netflix task. Similar uncertainties about the right form of output of models and prediction algorithms appear in most prediction tasks. Mainly I am interested here in assessing how many hidden variables are needed to generate ratings with a distribution close to the observed distribution of data.

We have two basic ways of modelling the output variable for the purpose of our prediction goal (to minimize RMSE). Either our model can generate ratings (we do not need a fully specified generative model $p(rating, user, movie, time)$ that generates the entire dataset – estimating $p(rating|user, movie, time)$ or sampling from that distribution is enough), or we can directly model the expected rating $E(rating|user, movie, time)$, to minimize the expected MSE loss. A model generating all five ratings, that proved effective in the Netflix Prize task, were Restricted Boltzmann Machines [Sal07a]. Most often a simplification was made, explicitly or implicitly treating discrete ratings as a Gaussian variable, with the mean modelled by a structure of hidden variables. In some realizations of the Gaussian approximation the posterior distribution of the Gaussian output was calculated explicitly, as when the model parameters were learned by MCMC, or approximated with Variational Bayes, but most popular in the Netflix task were neural-networks-

like methods that based on the cost function minimization, which can be understood as modelling the expected rating directly, without precisely specifying the posterior distribution of the output (without estimating the output variance). The expected value of the Gaussian approximation was usually clipped to the range $1-5$ [Fun06], or sometimes transformed through a sigmoidal link function to limit the range of the output variable [Pio09]. Typically used as a hidden structure matrix factorizations were sometimes modified by additional user-specific parameters, for example using a per-user scaling parameter [Kor09b, Pio09] or a transformation through a third degree polynomial [Pio09].

In this section I examine closer the issue – what is the right way to generate the output variable (ratings). Determining the proper way of output modelling is one step on the way to discovering which process plausibly generated the data. Creating a model for generating ratings is an intermediate goal for the task of minimizing expected MSE (I do not need to model accurately those aspects of the data that are unrelated to the chosen prediction task). The goal I set to myself is to build models capable of generating ratings with a possibly similar distribution to observed ratings, but using as few parameters as possible, because using too many parameters complicates the used methods, and often introduces overfitting the data, which can deteriorate the predictive accuracy. Looking at different possible kinds of outputs in efficient methods applied to the Netflix Prize task, we have several possibilities to verify: either to use a clipped and rounded normal variable with one hidden parameter (mean) or with two hidden parameters (mean and variance), or modelling each of the probabilities for ratings $1-5$ separately, with $4-5$ variables per rating. Close to the latter approach is treating the rating as an ordinal variable. It is also possible to model the output as a binomial variable, with one or two hidden parameters (mean and possibly range).

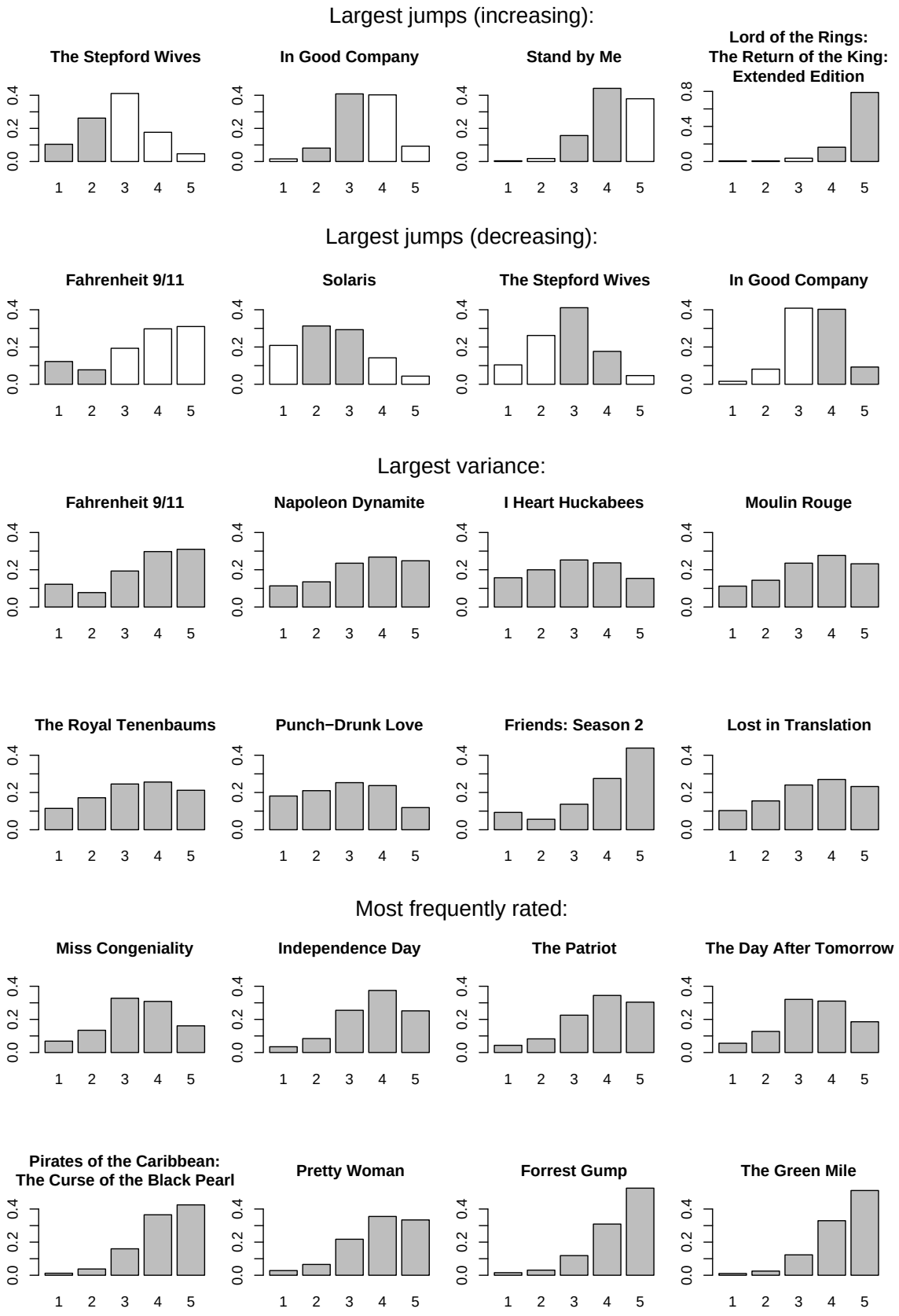
In the section I will visually examine empirical probability distributions for users and for movies. Then I will present experimental results on nonparametric models, that approximately generate the data. In those models the output is a rounded normal distribution, clipped to range $1-5$. Empirical distributions in the generated data will be compared with the observed data. I will conclude whether the clipped normal distribution is a good way to model the output, and whether it is enough to model the hidden mean, or is modelling the variance necessary. Further simplifying the methods is possible through modelling directly the conditional expected value $E(rating|user, movie, time)$.

We can wonder if the situation for simple models analyzed in this section is similar to more complex models that more accurately model the ratings. I have not verfied it – I assumed that for more complex models, with MSE accuracy smaller by $10-20\%$ ($10-20\%$ more variance explained), conclusions about the output variable would be similar. Of course it is possible that with a larger pattern modelling capability when using more parameters in the hidden structure, using more parameters for the output variable could give an accuracy improvement unnoticed in the simpler models.

Let's look first at the observed rating distributions for a few movies and users. Plot 18 shows the empirical rating probabilities of a few example movies chosen among the 1000 most frequently rated movies in the Netflix Prize dataset. "Empirical rating probabilities" denotes the estimated distribution of ratings in observed data, calculated here as maximum likelihood (ML) point estimators of probability of each rating for a given movie. The examples chosen are extreme: distributions with largest "jumps" (largest differences between adjacent ratings), and distributions with the largest variance. The examples were chosen among the 1000 most frequent movies, to make sure that the observed distributions are close to the real underlying distribution, and are not a result of random fluctuations. The bottom of the plot 18 shows the distributions of eight most frequently rated movies for comparison.

We conclude from the plot 18 that a Gaussian distribution with an appropriately

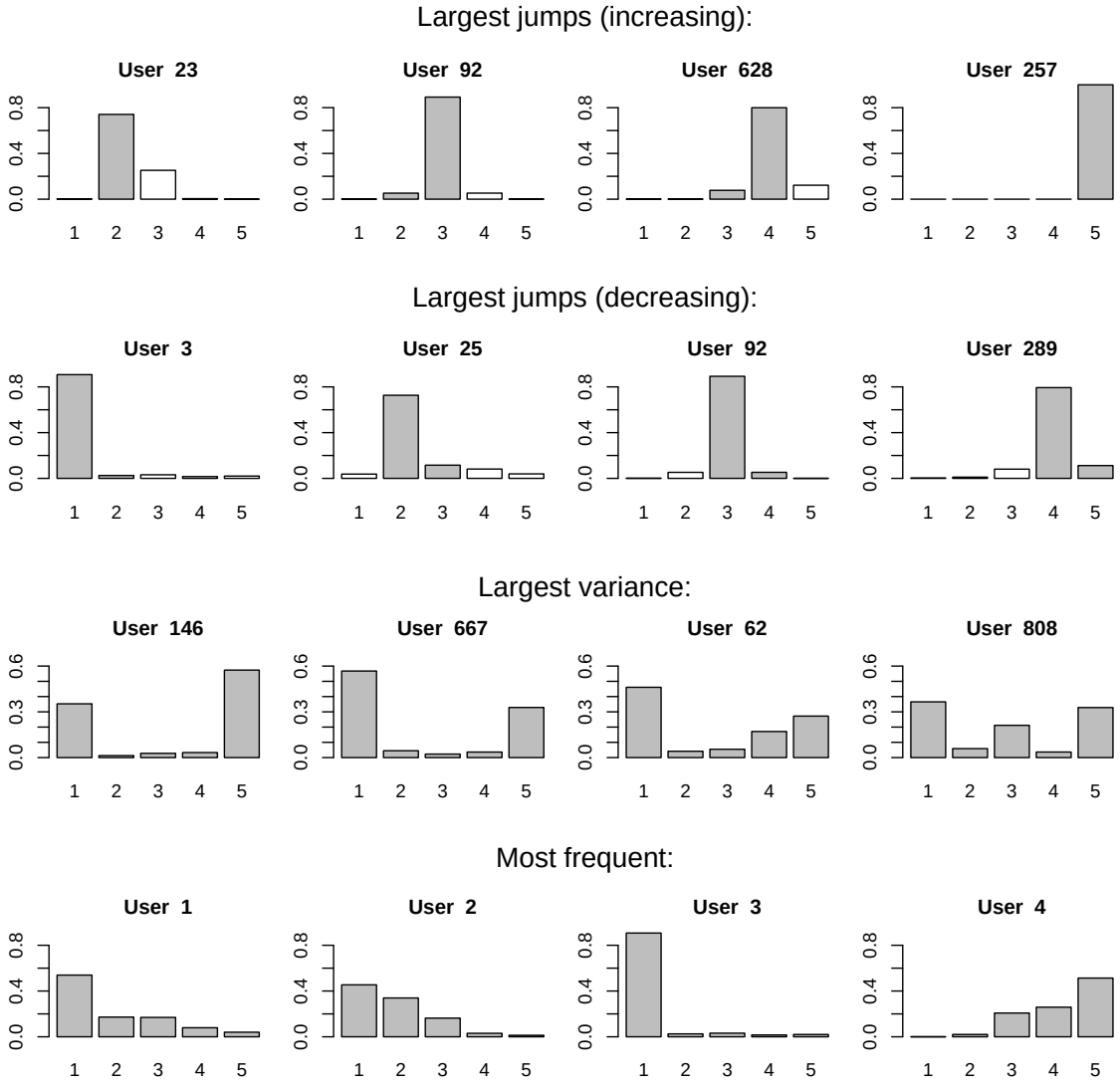
Figure 18: Movies (1k most frequent, min. support = 25773): example empirical rating probabilities.



chosen mean and variance, and clipped to range $1 - 5$, seems to be a good tool to model the rating distribution of a movie. The shown extreme distributions do not contain movies that visibly could not be modelled this way. A good choice seems to be also a binomial distribution or a mixture of binomial distributions.

Let's look at an analogous plot of the empirical probabilities for 1000 most frequent users in the dataset. Figure 19 shows the distributions of chosen users from that group: distributions with largest jumps, and most frequent users for comparison. Users are numbered according to their support. The smallest value of support (number of ratings) in the group of most frequent 1000 users is 2082. It is a value large enough to say that the visible characteristics of extreme distributions are not a result of randomness in the data, but have a real shape of the underlying user-specific distribution.

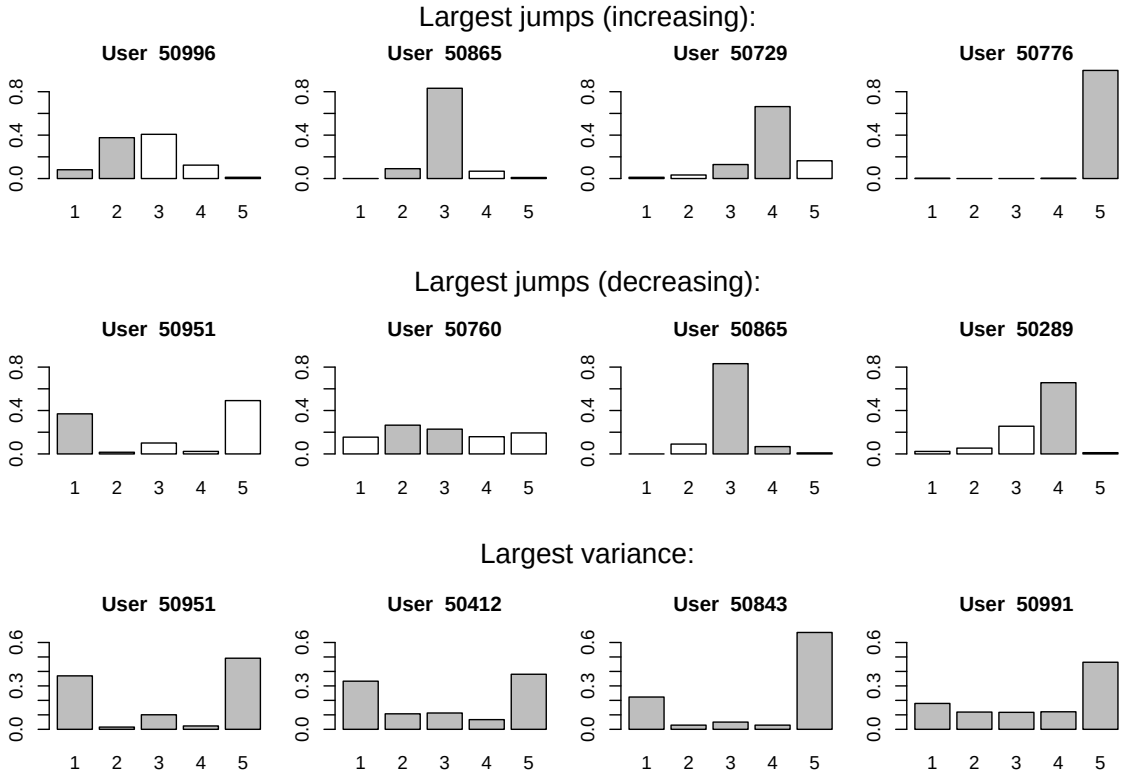
Figure 19: Users (1k most frequent, min. support = 2082): example empirical rating probabilities.



Users who give most ratings may be not representative for the whole set of users – they may have a different rating distribution than users who give less ratings. For example, it is likely that many of the users with many ratings give a fixed rating for movies never seen. Figure 20 shows the extreme empirical distributions in the group of users with the

frequency rank $50,000-51,000$. The smallest number of rated movies in this group is 517. We see that the extreme distributions in this group look similar to the previous group of most frequent users. The differences are small, in particular, there are no distributions with a large probability of rating 2.

Figure 20: Users (50k-51k support rank, min. support = 517): example empirical rating probabilities.

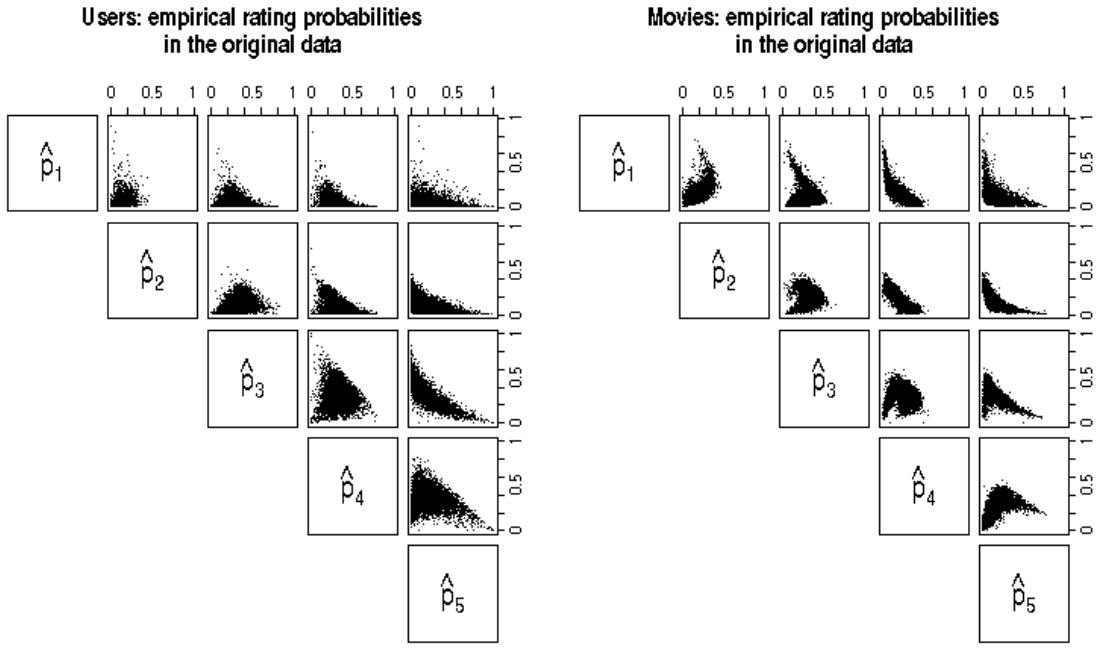


The plotted extreme empirical distributions for users (figures 19 and 20) differ considerably from the extreme distributions for movies (figure 18). The binomial distribution on the range $1-5$ fitted the movie distributions fairly well, but it cannot be used to model the user distributions. Clipped and rounded normal distribution could approximately generate the observed user distributions with an appropriate choice of variance: very small variance or very large. The overall conclusion from the plotted example distributions is that we should expect many user-specific patterns that visibly differ from typical one-parameter or two-parameter distributions. The capability of modelling such patterns may improve the accuracy. It is a probable reason for efficiency of the RBM method, which models each rating separately. Variants of RBM significantly improved accuracy of the ensemble of many efficient methods, so it is likely that they learn something that is not learned by other methods (most of the other methods included in the accurate ensembles used fewer parameters than RBM to model the output distribution).

Let's look at the plots visualizing much larger groups of users and movies. Figure 21 shows scatter plots of the empirical probabilities for a subset of $20,000$ users randomly drawn from $100,000$ users with most ratings, and analogous scatter plots for all $17,770$ movies. 2×10 scatter plots are presented, visualizing dependencies between empirical probabilities of each pair of ratings: $\hat{p}_i$ and $\hat{p}_j$, $i, j \in \{1, 2, 3, 4, 5\}$.

We approach the question: how to best model the output? Which probabilistic model

Figure 21: Empirical rating probabilities in the observed data – scatter plots



could generate similar data? Of course we could model the probability of each rating separately, using 5 parameters, and normalizing resulting probabilities to make them sum to 1, or using 4 parameters, where the probability of each rating is dependent on the four remaining ratings. But we see in figure 21 and on the plots of extreme distributions (figures 18, 19, 20), that there is a large degree of dependence of rating probabilities in each possible pair of ratings. The observed distributions are very distant from e.g. a Dirichlet distribution, where pairs of probabilities are close to being independent. The observed dependencies suggest that it is possible to model the output variable with fewer parameters than $4-5$ per rating. We want to reduce the number of parameters, because for two models with similar capability of explaining effects and dependencies in data usually the smaller model has better accuracy. The plots suggest, that a large part of the variability of the observed distributions can be explained by one location variable (for example, the mean of a normal distribution).

To verify to what extent the mentioned dimensionality reduction is possible, I developed four simple models for rating generation in a multitask learning setting with joint nonparametric priors for hidden parameters. The models are: hidden user mean, hidden movie mean, hidden user mean and variance, and hidden movie mean and variance. I will describe more in detail the user mean model. The remaining three are analogous.

The user mean model learns one parameter (mean) μ_i per user, and generates all user ratings using just this one parameter. It is a hierarchical model with multitask learning structure, with a common prior η for each user's hidden mean μ_i. Good practice in probabilistic modelling is to use a method with more parameters first, then after noticing a pattern, regularity, known distribution, attempt to reduce the model to a smaller or simpler one. We will use a nonparametric prior η in the form of a Dirichlet process, using a uniform distribution on a fixed grid of values as the base distribution (the only parameter of the Dirichlet process). "Nonparametric" means a model capable of increasing the number of its parameters with increased number of observations (unlike parametric ones, like Gaussian distribution, with a fixed number of parameters – mean and variance). In our setting this method essentially works as a histogram, but a classical histogram is calculated on observed data, and here we calculate it on unobserved variables, using the set

of observed samples as a density estimator.

The parameters of the model are learned iteratively, with an empirical Bayes type approach. The method is analogous to the nonparametric methods used later in sections 4.2.3 and 4.2.5. In the iteration K the prior $\hat{\eta}_{K-1}$ learned in the previous phase is used to draw samples from the posterior distribution for each μ_i, using the Metropolis-Hastings algorithm. We assume that the ratings are generated from a Gaussian distribution $N(\mu_i, \sigma^2)$, rounded to the nearest integer and clipped to range $1-5$, and this assumption is used to calculate the likelihood of observing the data, for example, the probability of giving the rating 1 by user i in this model is $p(D_i = 1|\mu_i) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{1.5} \exp(-\frac{1}{2\sigma^2}(x-\mu_i)^2)$. The parameter σ^2 was arbitrarily fixed to 1.1 in the user mean model and to 1.5 in the movie mean model. Then, taking the Empirical Bayes Monte Carlo approach, one sample from each user's posterior distribution is used for (inexact) maximum likelihood estimation of the common prior $\hat{\eta}_K$. On average $1/e \approx 36.8\%$ of samples would not be drawn when sampling with replacement. Because I did not want the generated values to disappear from the prior, I used sampling the set without replacement, which results in biased sampling. The algorithm starts with an initial prior $\hat{\eta}_0$ of $N = 100,000$ real values drawn from the uniform distribution on a range $< 0.5, 7.0 >$. In each subsequent iteration K the prior $\hat{\eta}_{K-1}$ is simply a set of N values drawn from the posterior distributions for μ_i in the $K-1$ -th iteration, one value per each user. There method was run for 200 iterations.

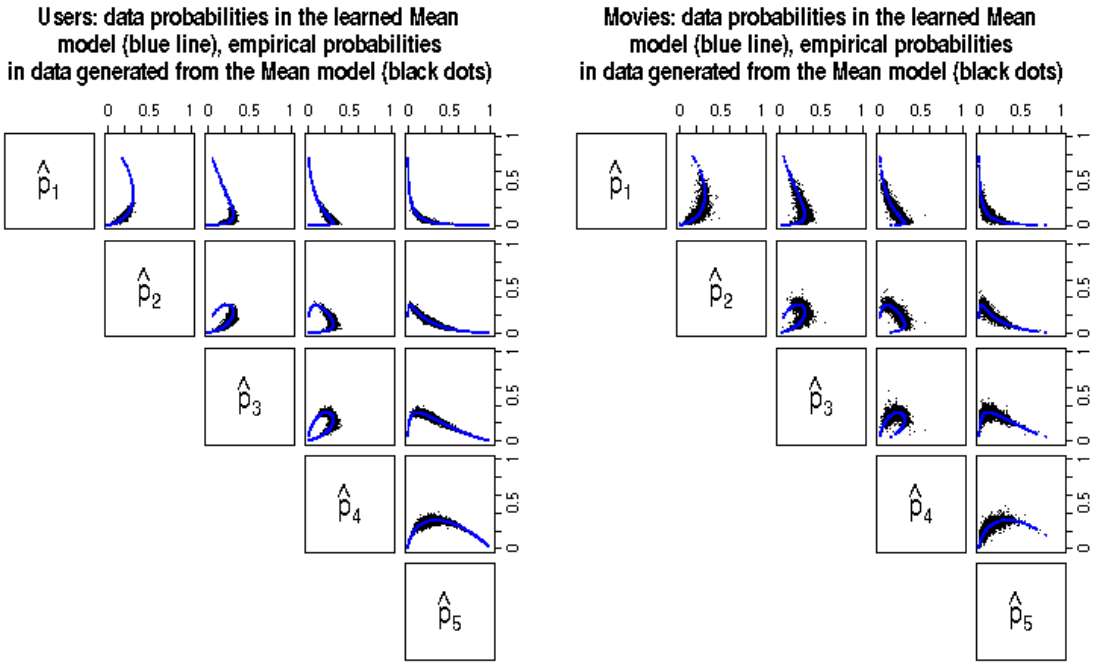
The model for a hidden movie mean is analogous, but it is more difficult to learn the nonparametric prior in it, because there are only $17,770$ tasks (in the user model there were $100,000$ tasks).

The algorithm worked well enough for the goals of the experiment (we want to discover the optimal shape of the prior, and observe how the model with this prior generates ratings), but it should be noted that the used form of the algorithm has its drawbacks. If some value disappears from the prior η_K in some iteration K, in the next iterations it will not appear again. As a result, the tails of the resulting prior distributions are far from being precise. A remedy for disappearing values could be drawing additional values in each iteration, for example, from a uniform prior (but this way we arbitrarily affect the shape of the learned prior). Another way is to increase the number of samples from the posterior distributions of μ_i, which form the prior η_K (in the used methods only one sample per user was used). Other, more "smooth" nonparametric priors could be used, in which near values are more probable to have a similar pdf value (comparing to commonly used nonparametric methods, our method was close to calculating a histogram, and we could use kernel density estimators instead).

Figure 22 shows scatter plots of the empirical probabilities (black dots) for the hidden mean model. The blue lines mark the probability, with which M_i ratings were drawn for each user i. On the left plots are marked $20,000$ users randomly chosen from the $100,000$ users with most ratings, for which the model was built. On the right plots are marked all $17,770$ movies in the hidden movie mean model. It is apparent that the empirical probabilities in the data generated by the learned Mean model have similar shapes as the empirical probabilities for the original data, shown on the earlier scatter plots (figure 21), but there are differences between these two groups of empirical distributions. The approximation by the Mean models is inaccurate, because having only one hidden parameter allows to change probabilities only along the fixed line (marked with the blue line on the plots).

A model with two parameters per rating: mean and variance, and a nonparametric prior S on the variance σ_i^2 turned out to better approximate the observed data. The prior distribution is learned like in the previous model, with empirical Bayes, using the Metropolis-Hastings algorithm to generate posterior samples from μ_i and σ_i^2. Figure 23 shows the empirical probabilities for data generated by the Mean+Variance model for users (left) and by the Mean+Variance model for movies (right). Visually, the plots for

Figure 22: Empirical rating probabilities in the data generated from Mean models – scatter plots



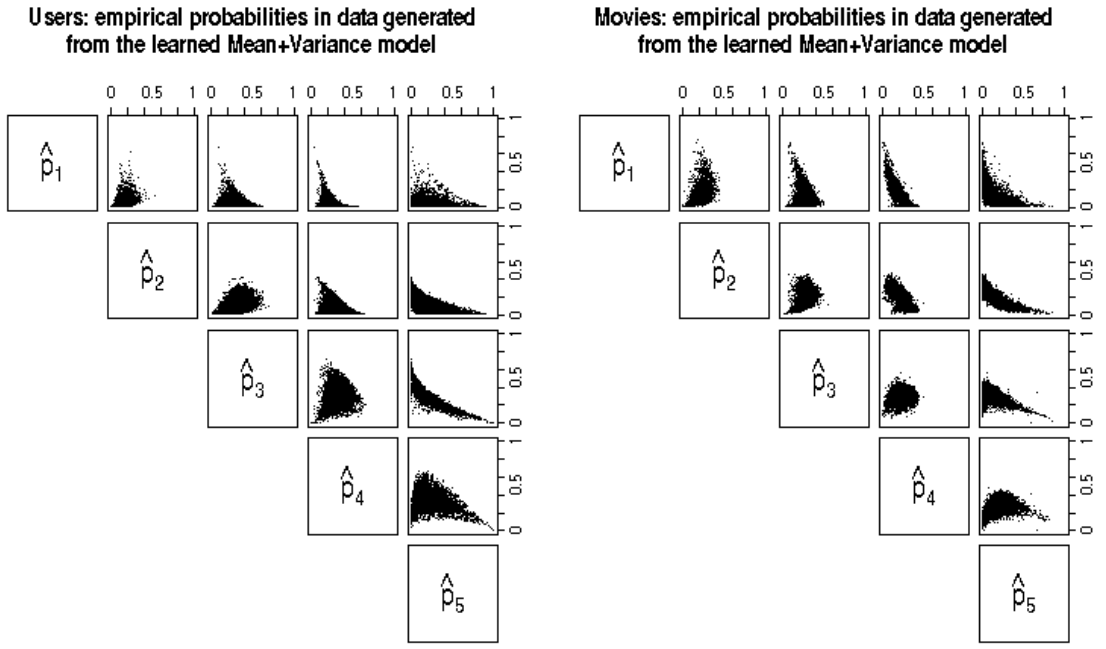
Mean+Variance model (figure 23) are much more similar to the original data (figure 21), than was the data generated from the Mean model (figure 22).

Let's examine plots visualizing the learned nonparametric priors η and S for the model Mean+Variance for users. Figure 24 visualizes the priors for mean and variance with histograms. Next are displayed: a QQ-plot that compares the learned prior for mean with quantiles from the Gaussian distribution, and a plot showing how the learned variance decreases with increasing number of ratings given by the user. Figure 25 displays an analogous set of plots in the model Mean+Variance for movies. The learned priors for mean in both models shown in figures 24 and 25 are similar to Gaussian distributions, and the priors for variance are similar to inverse Gamma distributions, but they are not exactly distributions from those families – there are differences visible. It should be noted that the shape of priors is more credible (likely closer to optimal priors) in the areas of large probability mass.

Summarizing the subject of output modelling in the Netflix task, the most accurate models described later in this work use only one parameter to model the expected output, so it can be suspected, that the number of parameters needed to generate ratings can be reduced to less than 4 needed to model the probability of each rating separately. This section explored the idea to use clipped and rounded Normal distribution to generate ratings. Two models were proposed: the Mean model and the Mean+Variance model, each in two versions – for users and for movies. We plotted the empirical probabilities of data generated from the learned four models. Comparing with the empirical probabilities in original data, the data generated from Mean model visually differs from the original data – the generated data has similar shape, but concentrates only in a certain region. The Mean model is not capable of modelling fully the original data. In turn, the data generated from the Mean+Variance model is visually very similar to the original data. I conclude, that one hidden parameter is not enough to generate output similar to the original ratings, but two hidden parameters allow for a good approximation, hence we can reduce the dimension of the output variable.

Instead of using a Gaussian distribution, it is possible tu use a binomial distribution

Figure 23: Empirical rating probabilities in the data generated from Mean+Variance models – scatter plots



[Wu09], but in that case useful would be a possibility of increasing variance, for example, by treating the location parameter as a random variable with uncertainty. A binomial distribution with increased variance would be good enough for modelling movie distributions, but is not capable to fully model user distributions. For some users decreasing the variance is needed, for example, limiting the range of binomial distribution to shorter than $1-5$ could be used. For modelling users with a large variance, e.g. users who give many ratings 1 and 5, a mixture of binomial distributions could be used.

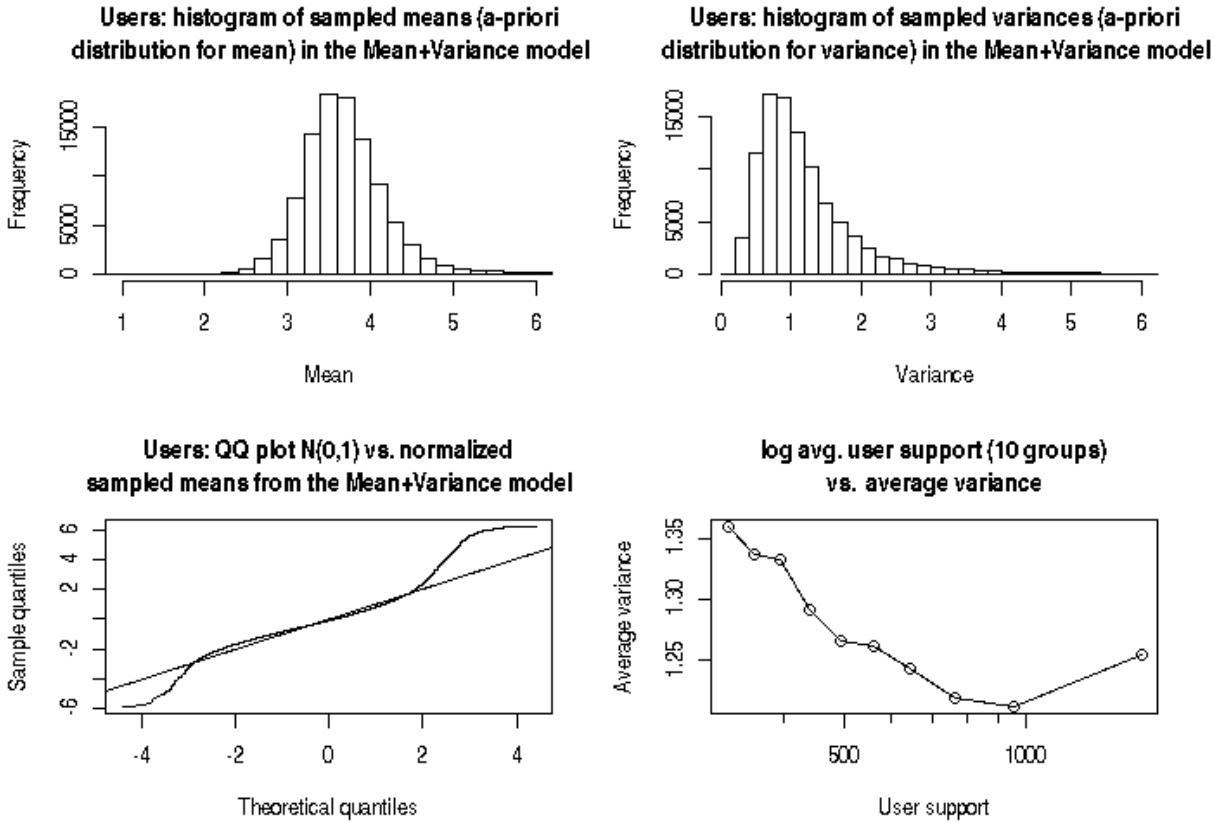
Of course, in the Netflix Prize task we are mainly interested in optimizing the predictive accuracy according to the MSE loss, so we do not need an accurate model for generating ratings here, but it is enough to model well the expected output $E(rating|user, movie, time)$. Modelling the output variance parameter $\sigma^2(user, movie)$ [Tom07] can useful for our prediction task – it influences weighting of observations and can improve predictive accuracy (see section 4.3.3). Approximating the posterior variance (note that this is different from assessing the σ^2 parameter in the model) or the full posterior distribution of ratings is needed to calculate recommendations (see section 3.4).

Another possibility, not tried here, is treating ratings as ordinal variables [Ste09, Paq10, Kor11], allowing to obtain better RMSE accuracy than modelling just the expected value of a Gaussian variable [Paq10, Kor11]. Still, it would be useful to determine if a four-parameter ordinal variable is better than a two-parameter truncated Gaussian variable, for the Netflix data.

4.2.3 Only biases

A very common component of predictive models are biases, that is parameters that model systematic constant deviations of a predicted response variable. A bias variable can be global for all data, or can model the deviation for individual objects or groups of objects, observations, features, time period, location, etc. Biases explain a large proportion of explainable variability in the prediction task Netflix Prize, and for that reason it is useful to examine them closer, to make sure that they are modelled in the right way. For many

Figure 24: Learned prior distribution in the Mean+Variance model for users: histograms of prior for mean and prior for variance, QQ plot mean vs. Gaussian, variance vs. log user support



datasets, simple models containing biases give prediction close to the optimal (for example, for the two tasks of modelling match outcomes, described in section 6.3). An advantage of bias models is that simple cost function optimization methods are often sufficiently accurate for parameter inference.

In this section I will examine some of the simplest models for the Netflix Prize task – models containing only movie biases, user biases and the global mean. The models of biases will be a good example illustrating the methodology of prediction used in this work. While developing different predictive models we are faced with similar questions, and the answers are more clear for simple models. I will compare different approaches to prediction and different ways of learning parameters. Simultaneous learning of biases with regularization turned out to be a good preprocessor or a good part of other methods, such as regularized SVD with biases.

The output variable will be approximated with a Gaussian distribution, as it was done in most methods used for the Netflix task. This decision was to some extent justified in the previous section.

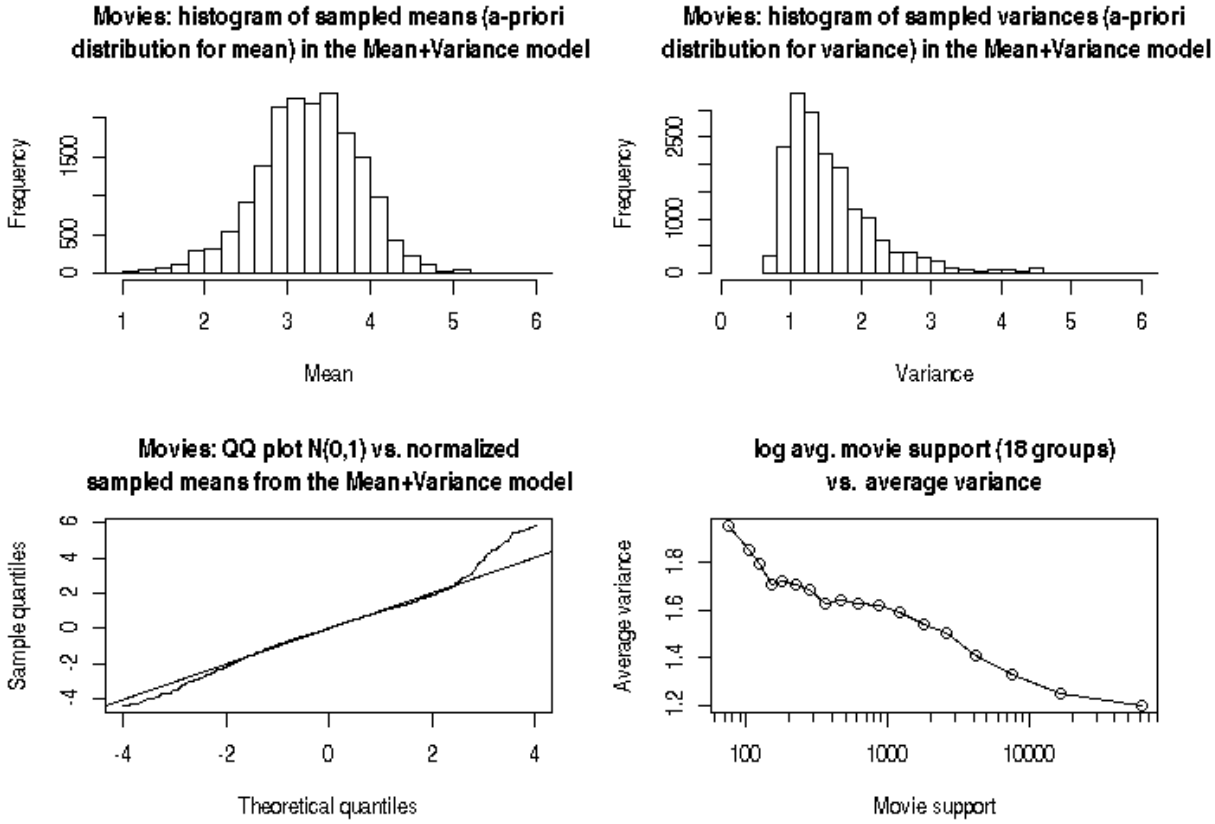
The analyzed models with biases have the form:

$$\hat{r}_{ij} = \mu + c_i + d_j$$

where c_i is the bias of user i, d_j is the bias of movie j, and $\mu = 3.6033$ is a constant – global mean of the training set.

Accuracy is measured by RMSE_{probe}, that is RMSE on the whole probe set. The predictions of the model were clipped to range $1 - 5$ before calculation of RMSE. The

Figure 25: Learned prior distribution in the Mean+Variance model for movies: histograms of prior for mean and prior for variance, QQ plot mean vs. Gaussian, variance vs. log user support



results of all compared methods are shown in table 7.

First I have to explain the general approach to prediction taken here. We mentioned in section 2.1, that the preferred approach to prediction is building a possibly most accurate probabilistic model for data, inference of the posterior distribution, and estimation with decision theory. Of course when working with real-world data, I do not know the exact probabilistic model, from which the data comes. Even with a known probabilistic model, often computing the exact inference is complicated or has too large computational complexity (this is not the case of the biases model, but it is the case in other methods). For these reasons, the probabilistic approach will not be applied precisely. I will examine the parameters learned in a method, check whether assumptions taken are met, and aim to move away from the Bayesian approach towards simplified approaches, such as neural networks.

Let's describe one by one the methods listed in table 7. Global mean is simply the average rating μ over the training set:

$$\mu = \frac{1}{N_1} \sum_{ij \in Tr} r_{ij} = 3.6033, \ N_1 = |Tr| = 99,072,112$$

RMSE of the global mean is put in the table for comparison, as a baseline to show how much variability in data was explained by the models of biases. The subject of predicting with one global mean variable was discussed in section 4.2.1, along with the arguable use of the average rating and discussing the difference between the average rating on the training and on the probe set.

Table 7: Models with biases only. Comparison of experimental results.

Method	Parameters	Equation	Iterations	RMSE
global mean		μ	1	1.1296
only user bias, MAP	$\lambda_c = 8$	$\mu + c_i$	1	1.0638
only movie bias, MAP	$\lambda_c = 25$	$\mu + d_j$	1	1.0527
biases alt. MAP, sequential, user-movie	$\lambda_c = 8, \lambda_d = 25$	$\mu + c_i + d_j$	1	0.9884
biases alt. MAP, sequential, movie-user	$\lambda_c = 8, \lambda_d = 25$	$\mu + c_i + d_j$	1	0.9844
biases alt. MAP, simultaneous	$\lambda_c = 0, \lambda_d = 0$	$\mu + c_i + d_j$	3	0.9870
biases alt. MAP, simultaneous	$\lambda_c = 8, \lambda_d = 25$	$\mu + c_i + d_j$	3	0.9828
biases alt. MAP, simultaneous, opt.	$\lambda_c = 7, \lambda_d = -3$	$\mu + c_i + d_j$	3	0.9826
biases Bayesian, alt. expectation		$\mu + c_i + d_j$	4	0.9829
biases Bayesian, Gibbs sampl., expectation		$\mu + c_i + d_j$	50 + 1	0.9829
biases Bayesian, Gibbs sampl., avg. of samples		$\mu + c_i + d_j$	50 + 50	0.9831
biases Bayesian, nonparam. prior, avg. samples		$\mu + c_i + d_j$	50 + 50	0.9860

The global mean, a component of all models in this section, will be treated as a fixed constant $\mu = 3.6033$. Treating the global mean as an additional variable with a proper Bayesian inference would have no significant influence on prediction accuracy.

The model of user bias for simplicity is written as $\mu + c_i$ for user i. This notation is ambiguous, as it does not specify the precise meaning of the parameter c_i. Model structures specified this way will be treated either in a probabilistic way, where c_i is a random variable with a prior distribution, or in a cost function minimization approach, where c_i is a single-valued parameter with value found through minimizing a regularized cost function. For simple bias models in this section, the Bayesian approach with proper choice of prior distributions results in similar predictive accuracy to cost function minimization with proper choice of regularization.

In the Bayesian treatment of the user bias model all parameters c_i (one for each user) are random variables, where we assume equal prior distributions $N(0, \tau_c^2)$ (equal, because we assume exchangeability of users). About the ratings r_{ij} I assume, simplifying, that they were sampled from a normal distribution $N(\mu + c_i, \sigma^2)$. I assume here that τ_c^2 and σ^2 are chosen constants. We are minimizing expected MSE, and the resulting optimal prediction is the expected value of the posterior rating distribution: $Er_{ij} = \mu + Ec_i$. The Ec_i values have the form of shrinkage estimators with a regularization parameter $\lambda_c = \frac{\sigma^2}{\tau_c^2}$:

$$Ec_i = \frac{1}{|J_i| + \lambda_c} \sum_{j \in J_i} (r_{ij} - \mu)$$

The constant λ_c was set to the integer value that results in minimal RMSE on the test set. The chosen best integer value was $\lambda_c = 8$. It is possible also to learn λ_c jointly from all tasks (learning of $480,189$ parameters can treated as a set of similar tasks) – the approach of multitask learning (or empirical Bayes [Car00, Nor07]) will be used in some other models in this section.

The movie bias model $\mu + d_j$ is similar. Predictions are made by $Er_{ij} = \mu + Ed_j$, where d_j is a movie bias. The expected values Ed_j have an analogous form of shrinkage estimators:

$$Ed_j = \frac{1}{|I_j| + \lambda_d} \sum_{i \in I_j} (r_{ij} - \mu)$$

As before, we set λ_d to an integer value minimizing RMSE on the test set (the probe set). For the movie bias model it is $\lambda_d = 25$.

Let's move to the models with both biases, user bias c_i and movie bias d_j. All of these models have the form $\mu + c_i + d_j$, where $\mu = 3.6033$ is a constant, c_i are $480,189$ variables, one for each user, and d_j are $17,770$ variables, one for each movie. The simplest idea of learning the parameters, which worked well as a preprocessing for more complex methods [Fun06, Bel07a], was sequential learning: first calculate the movie biases, then calculate the user biases on the resulting residuals (the method labeled in table 7 as movie-user):

$$\hat{d}_j = \frac{1}{|I_j| + \lambda_d} \sum_{i \in I_j} (r_{ij} - \mu) \qquad \hat{c}_i = \frac{1}{|J_i| + \lambda_c} \sum_{j \in J_i} (r_{ij} - \mu - \hat{d}_j)$$

Reversed sequence of learning leads to worse accuracy. In the model listed in table 7 as user-movie first the user biases are calculated, then the movie biases on the residuals:

$$\hat{c}_i = \frac{1}{|J_i| + \lambda_c} \sum_{j \in J_i} (r_{ij} - \mu) \qquad \hat{d}_j = \frac{1}{|I_j| + \lambda_d} \sum_{i \in I_j} (r_{ij} - \mu - \hat{c}_j)$$

In place of sequential learning, better predictive accuracy gives repeating several times learning user and movie biases on residuals of each other – a method labeled in table 7 as "MAP simultaneous":

$$\hat{c}_i = \frac{1}{|J_i| + \lambda_c} \sum_{j \in J_i} (r_{ij} - \mu - \hat{d}_j) \qquad \hat{d}_j = \frac{1}{|I_j| + \lambda_d} \sum_{i \in I_j} (r_{ij} - \mu - \hat{c}_j)$$

Three different sets of parameters were used for simultaneous learning. The first version without regularization: $\lambda_c = \lambda_d = 0$, gives much worse results than the regularized versions. The second set of regularization parameters was optimal for models "only user bias" and "only movie bias": $\lambda_c = 8$, $\lambda_d = 25$. The third set $\lambda_c = 7, \lambda_d = -3$ was learned by an automatic optimizer, to minimize RMSE on test set (probe set). The optimizer used was the Praxis module from the library Fortran Netlib, which was also used for several methods described further in the work. The parameters after optimization were rounded to the nearest integer values. Interestingly, the optimal parameter λ_d turned out to be negative. Automatic tuning resulted in $\mathrm{RMSE}_{probe} = 0.9826$, the lowest of the learning methods examined in this section.

I gave several examples, how is it possible to choose the regularization parameters λ_c, λ_d by minimizing the validation error. If we specify a full probabilistic model with shared hyperpriors, then we can estimate these regularization parameters from the training data in an empirical Bayes approach. If the probabilistic model is correct (is close to the unknown model that generated the data) and if we have not oversimplified the parameter inference, accuracy on the test set should be close to the results of automatic parameter tuning.

We have decided on the basic form of the model – that predictions will be made by the sum $\mu + c_i + d_j$. If c_i and d_j are treated as random variables, different approaches can be used to infer their posterior distributions. One possibility is to use MCMC methods based on Gibbs sampling, where parameters θ_i are cyclically sampled from their posterior distributions conditioned on earlier drawn values of all remaining parameters $p(\theta_i | D, \boldsymbol{\theta}_{-i})$. This way samples from the joint distribution $p(\boldsymbol{\theta}|D)$ are obtained, although the subsequent samples are not independent. Another method is the Variational Bayes approximation, where the parameters are split into groups, about which is assumed, that the groups are independent in the joint posterior distribution. With this constraint, minimizing the Kullback-Leibler divergence between the approximation and the true posterior using variational methods leads to approximating the posterior distribution of θ_i with: $\log\ \hat{p}(\theta_i | D, \boldsymbol{\theta}_{-i}) = C + E_{\hat{p}(\boldsymbol{\theta}_{-i}|D)} \log\ p(\boldsymbol{\theta}|D)$. An advantage of the VB method in comparison to MCMC is much fewer iterations (because the end result of VB are probability distributions, unlike MCMC, which needs to average over generated samples), and often faster convergence.

Yet another possibility is to use alternating MAP estimation of parameters. That approach worked well for the simple model of two biases (the already mentioned method "MAP simultaneous"), but, as we shall see later in the work, it has worse accuracy than MCMC and VB in more complex models (although e.g. for SVD models accuracy is largely improved by specially chosen priors, unequal for all users or for all movies).

I will examine the Variational Bayesian method and MCMC based on Gibbs sampling for our model of biases. The VB method reduces here to alternating estimation of c_i and d_j with their expected values, with learning the joint priors in the hierarchical model. I assume a joint prior distribution of c_i in the form of $N(\overline{c}, \tau_c^2)$, and an analogous prior distribution of d_i as $N(\overline{d}, \tau_d^2)$. To simplify, I will use maximum likelihood estimation for the parameters τ_c^2, τ_d^2, σ^2, that is MAP estimation (maximum a-posteriori) with flat priors (the parameters τ_c^2 and τ_d^2 can be called hyperparameters, and their prior distributions – hyperpriors).

The user and movie parameters depend on each other, and learning them will require multiple iterations. We are interested in the distribution $p(c_i|D, \mathbf{d}, \tau_c^2, \sigma^2)$, where d_j's are either drawn from their current posterior distributions in the MCMC approach with Gibbs sampling, or fixed to their expected value Ed_j of the distribution $p(d_j|D, \mathbf{c}, \tau_d^2, \sigma^2)$ in the VB approach. I assume here that τ_c^2, τ_d^2 and σ^2 are known constants, calculated based on the approximated distributions or samples of c_i and d_j in previous iterations. The conditional posterior distribution $p(c_i|D, \mathbf{d})$ is Gaussian:

$$p(c_i|D, \mathbf{d}) \propto p(D|c_i, \mathbf{d})p(c_i) = f(c_i) = \exp\Big(C_1 - \frac{\sum_{j \in J_i}(r_{ij} - \mu - d_j - c_i)^2}{2\sigma^2} - \frac{(c_i - \overline{c})^2}{2\tau_c^2}\Big)$$

$$= \exp\Big(C_2 + c_i * \Big(\frac{\sum_{j \in J_i}(r_{ij} - \mu - d_j)}{\sigma^2} + \frac{\overline{c}}{\tau_c^2}\Big) - \frac{1}{2}c_i^2 * \Big(\frac{|J_i|}{\sigma^2} + \frac{1}{\tau_c^2}\Big)\Big)$$

With fixed $\mathbf{d}$ (sampled d_i in MCMC, or d_i fixed to Ed_i in VB) the posterior distribution $p(c_i|D, \mathbf{d})$ is $N(Ec_i, Var\ c_i)$ with the following parameters:

$$p(c_i|D, \mathbf{d}) = \frac{1}{\sqrt{2\pi Var\ c_i}} \exp\Big(-\frac{(c_i - Ec_i)^2}{2\ Var\ c_i}\Big)$$

$$Var\ c_i = \frac{1}{|J_i|/\sigma^2 + 1/\tau_c^2} \tag{10}$$

$$Ec_i = \Big(\frac{\sum_{j \in J_i}(r_{ij} - \mu - d_j)}{\sigma^2} + \frac{\overline{c}}{\tau_c^2}\Big) * Var\ c_i \tag{11}$$

I assume equal prior distribution $N(\overline{c}, \tau_c^2)$ for all c_i. The parameters $\overline{c}$, τ_c^2 of the prior distribution are point estimated with empirical Bayes method, on the basis of the posterior distributions of c_i's in the previous iteration of the algorithm.

$$\overline{c} = \frac{1}{N}\sum_{i=1}^{N} Ec_i \tag{12}$$

$$\tau_c^2 = \frac{1}{N}\sum_{i=1}^{N}((Ec_i)^2 + Var\ c_i) - \overline{c}^2 \tag{13}$$

Analogously, the posterior distribution $p(d_j|D, \mathbf{c})$ with fixed c_i's is Gaussian $N(Ed_j, Var\ d_j)$:

$$Var\ d_j = \frac{1}{|I_j|/\sigma^2 + 1/\tau_d^2} \tag{14}$$

$$Ed_j = (\frac{\sum_{i \in I_j}(r_{ij} - \mu - c_i)}{\sigma^2} + \frac{\overline{d}}{\tau_d^2}) * Var\ d_j \tag{15}$$

The prior distribution of d_i was assumed in the form $N(\overline{d}, \tau_d^2)$. The parameters $\overline{d}$, τ_d^2 are calculated as point estimates:

$$\overline{d} = \frac{1}{M}\sum_{j=1}^{M} Ed_j \tag{16}$$

$$\tau_d^2 = \frac{1}{M}\sum_{j=1}^{M}((Ed_j)^2 + Vard_j) - \overline{d}^2 \tag{17}$$

The parameter σ^2 can be estimated with maximum likelihood, using sampled c_i and d_j:

$$\sigma^2 = \frac{1}{|Tr|}\sum_{ij \in Tr}(r_{ij} - \mu - c_i - d_j)^2 \tag{18}$$

If we assume independence of the two groups of parameters c_i and d_j in the joint posterior distribution, then $Ec_i d_j = 0$, and maximum likelihood point estimation of σ^2 is following:

$$\sigma^2 = \frac{1}{|Tr|}\Big(\sum_{ij \in Tr}(r_{ij} - \mu - Ec_i - Ed_j)^2 + Var\ c_i + Var\ d_j\Big) \tag{19}$$

The MCMC method with Gibbs sampling is simpler conceptually comparing to Variational Bayes, has similar predictive accuracy, but needs many more iterations of the algorithm. The MCMC method iterates between sampling c_i according to (10),(11), sampling d_j according to (14),(15), and re-estimating the hyperparameters $\overline{c}$, $\overline{d}$, τ_c^2, τ_d^2, σ^2 according to (12),(13),(16),(17),(18) Table 7 compares two implementations. In the first one predictions are made by the expected value $Er_{ij} = \mu + Ec_i + Ed_j$ after 50 iterations of Gibbs sampling. In the second one predictions are made by running the method for additional 50 iterations and averaging the samples (averaging expectations leads to the same accuracy as the first method). After $50 + 50$ iterations the following parametrization of the common prior distributions was obtained: $\overline{c} = 0.07, \overline{d} = -0.30, \tau_c^2 = 0.19, \tau_d^2 = 0.26, \sigma^2 = 0.84$, which corresponds to the regularization coefficients $\lambda_d = \sigma^2/\tau_c^2 = 4.42$ i $\lambda_d = \sigma^2/\tau_d^2 = 3.23$.

The Variational Bayesian method assumes independence of c_i and d_j in the posterior distribution. Posterior c_i is approximated by $\log\ \hat{p}(c_i|D, \mathbf{d}) = C + E_{\hat{p}(\mathbf{d}|D)} \log\ p(c_i, \mathbf{d}|D)$, where $\hat{p}(\mathbf{d}|D)$ is the approximation of the probabilities $p(d_j|D, \mathbf{c})$ calculated in the previous iteration of the algorithm. In the model of biases the VB method boils down to putting Ed_i in place of d_i's in the equation (11), and putting Ec_i in place of c_i's in the equation (15), hence it is called "alternating expectation" in the table 7. This method needed four iterations to converge, with $\text{RMSE}_{probe} = 0.9829$. The algorithm iterates between calculating approximate posterior c_i and d_j distributions and re-estimating hyperparameters:

```
1.  for (iter in 1..4)
2.    for (i in 1..480189) // loop over users
3.      update c_i according to (10),(11)
4.    for (i in 1..17770) // loop over movies
5.      update d_j according to (14),(15)
6.    update c̄, d̄, τ_c^2, τ_d^2, σ^2 according to (12),(13),(16),(17),(19)
```

Comparing the accuracy of the presented parametric methods, we can wonder why the method "MAP simultaneous, opt." gave the best accuracy. A probable reason is inaccuracy of the assumed model.

The last method described are biases with nonparametric priors. A question arises, whether Gaussian distribution is the right choice for a prior distribution of parameters c_i, d_j, and if not, whether it is worth to model the deviation accurately. To verify if the choice of Gaussian for priors is correct, I assumed a prior in a nonparametric form of a Dirichlet process. We have enough tasks sharing the same prior (over 480 thousand c_i's and over 17 thousand d_j's), to make such an attempt of nonparametric modelling. The used nonparametric method is similar to the described in [Yu04, Tre04, Xue07]. I used similar methods in sections 4.2.2 and 4.2.5

The nonparametric model is formed by changing the priors $N(\overline{c}, \tau_c)$ and $N(\overline{d}, \tau_d)$ to Dirichlet processes with concentration parameter zero (i.e. without choosing a hyperprior probability measure). A Dirichlet process (DP) is a generalization of the Dirichlet distribution to infinite (of cardinality continuum) number of parameters, and can be understood as a distribution over probability distributions (probability measures). In the used model there are no hyperparameters $\overline{c}$, $\overline{d}$, τ_c, τ_d. Still remains the parameter σ^2, which exact choice has little influence on the result of the algorithm, and was set to $\sigma^2 = 0.5$.

A simplification made is assuming a grid of possible values. Our prior distribution in the form of a DP generates probability distributions over the points of the grid, one distribution for every task (for one DP prior the tasks are users, for the second prior, the tasks are movies). The assumed grid of values for both priors are uniformly spaced 500 points on the range $< -5, 5 >$, in increments of 0.02. The method can be seen as repeatedly using a histogram to estimate the density of priors for hidden variables. The nonparametric method used here and also in other places of the work (output modelling, priors for SVD with one feature) has its disadvantages, such as unclear behavior for tails, but the method was good enough for visualization and to assess what would be a good parametric form of the priors.

There are multiple ways to realize the algorithm with DP priors. I decided on an approach where the values of c_i and d_j are sampled from their posterior distributions using the Metropolis-Hastings algorithm [Tar05, Mos95]. Likelihood of c_i depends on values of d_j, and vice-versa, so I use here the Gibbs sampling method, alternating using sampled d_j to sample c_i, and using sampled c_i to sample d_j. We can say, that in this nonparametric method the set of posterior samples in one iteration is the prior distribution in the next iteration.

Instead of sampling with replacement and obtaining independent samples from the prior, I used sampling without replacement (realization through the function random_shuffle). If e.g. in the prior distribution some value appears 10 times, it will appear exactly 10 times in the set after one sample from proposal distribution of each user (or movie).

The pseudocode of the algorithm is following (the numeric constants are left unnamed, to save space and improve clarity):

```
1.  Initialize c and d to random values from
    the discrete set [−5, −4.98, −4.96, −4.94, ..., 4.98, 5]
2.  for (iter in 1..100)
3.    c^new = random_shuffle(c)
4.    for (i in 1..480189) // loop over users
5.      l1 = log_likelihood_c(c_i, J_i)
6.      l2 = log_likelihood_c(c_i^new, J_i)
7.      if (exp(l1 − l2) > random())
8.        c_i = c_i^new
9.    d^new = random_shuffle(d)
10.   for (j in 1..17770) // loop over movies
11.     l1 = log_likelihood_d(d_j, I_j)
12.     l2 = log_likelihood_d(d_j^new, I_j)
13.     if (exp(l1 − l2) > random())
14.       d_j = d_j^new
15. c̄ = average c from iterations >= 50
16. d̄ = average d from iterations >= 50
```

The function random_shuffle($\mathbf{c}$) returns a random permutation of vector $\mathbf{c}$, each permutation with the same probability. The function random() returns a sample from the uniform distribution $U(0,1)$.

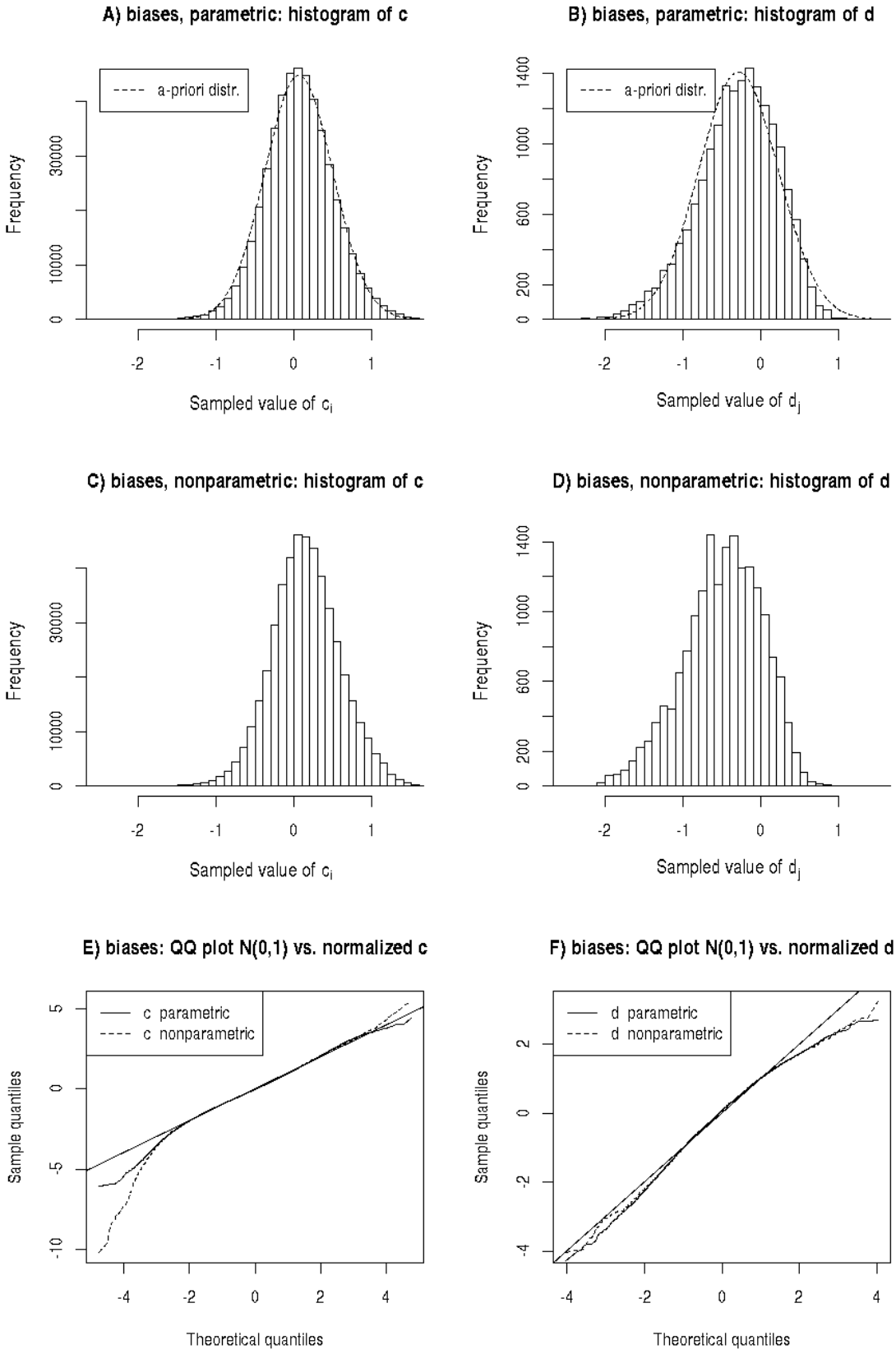
The function log_likelihood_c(x, J_i) returns the logarithm likelihood of $c_i = x$, given the observed data, assuming that $\mathbf{d}$ parameters are fixed. Because we assume, that the data is from distribution $N(\mu + c_i + d_j, \sigma^2)$, log_likelihood_c($x$, J_i) $= \frac{1}{2\sigma^2} \sum_{j \in J_i} (r_{ij} - \mu - x - d_j)^2$. I assumed $\sigma^2 = 0.5$ (a more accurate choice of σ^2 does not make a visible difference to the results). In the implementation it is necessary to prevent overflows of the function $\exp(l1 - l2)$, for example, by clipping $l1 - l2$ to the range $[-10, 10]$.

Now we'll visually examine learned prior distributions in MCMC models with Gibbs sampling with a parametric prior (Gaussian distribution) and nonparametric prior (Dirichlet process). Six plots in figure 26 visualize the learned priors. The plots A) and B) are histograms of samples c and d coming from the learned model of biases with Gaussian priors. For comparison, the prior Gaussian distributions were marked with a dashed line. The plots C) and D) are analogous histograms for samples of c and d in the nonparametric method. E) and D) are quantile-quantile plots, on that the distributions of c and d obtained from the parametric and nonparametric methods, after normalizing (centralization and dividing by sample standard deviation) are compared with the standard normal distribution.

We see on the plots that there are places where the learned prior distributions differ from a Gaussian distribution. The most visible differences are that the distribution of c on the negative side has a longer tail than a Gaussian, and in turn the distribution of d on its positive side has a shorter tail than a Gaussian. The second observation is that no large difference is visible in the posterior distributions between using a parametric Gaussian prior and a nonparametric prior, therefore a correction of the Gaussian assumption should not make a large difference (but the automatic optimization of regularization parameters suggests, that it is possible to improve something, maybe another shape of prior for movies would make a difference – this possibility was not investigated).

Let's sum up the simplifications made in the used methods. I assumed shared Gaussian priors for c_i's and d_j's, and treated the output rating as a Gaussian variable (the output choice was discussed in section 4.2.2). I assumed no important difference between the training and test sets (issue mentioned in section 4.2.1). I assumed exchangeability of users and exchangability of movies, and I applied an approach, which can be understood as hierarchical Bayesian modelling, multitask learning, or empirical Bayes (the boundaries of those techniques overlap). The assumption of exchangeability (equal prior) is inaccurate – we saw in section 3.5 that the average rating decreases with user frequency, but increases with movie frequency. The parameters of the priors (hyperparameters) were point

Figure 26: Learned priors of c, d; in the parametric and nonparametric method



estimated in the Empirical Bayes setting by maximum likelihood, instead of, for example, choosing hyperpriors and inferring the posterior distribution of the hyperparameters. Similarly, the parameter σ^2 was point estimated by maximum likelihood. I assumed equal variance σ^2 for all ratings. Accuracy can be improved by modelling variance individually for users and movies [Tom07]. A small simplification was that the VB method assumed independence of groups of parameters c_i from d_j in the joint posterior distribution. There were also many simplifications in the used nonparametric method.

It turned out that the best prediction accuracy was obtained by alternating MAP estimation with weights optimized with automatic parameter tuning ("MAP simultaneous opt."), which gave $\text{RMSE}_{probe} = 0.9825$. This method is capable of correcting some inaccuracies of the chosen probabilistic model, but the probabilistic methods were not much worse: alternating expectation and two Gibbs sampling methods gave RMSE_{probe} between 0.9829 and 0.9831. A similarly good result, $\text{RMSE}_{probe} = 0.9828$ ("MAP simultaneous") was obtained by using regularization parameters λ from single bias models, thus optimal for other (similar) models. Summarizing, all five variants of parametric methods with simultaneous learning of biases turned out to have good accuracy here.

4.2.4 Which movie is the best?

I took a preliminary look at the data (sections 3.3, 3.5), described the first simple models (section 4.2.3), and earlier I listed the most common issues encountered while building a recommender system (section 3.1). Now let's reflect on how to use the methods minimizing RMSE to calculate lists of recommendations for users. Simple models based only on biases do not give us yet a possibility to obtain personalized recommendations, but they let us consider one global list of recommendations – a list of top items common for all users. Lists of global top items or top items within a group of items are often a desirable feature on sites collecting opinions about items, e.g. in the form of ratings. In particular, in recommender systems lists of top items are a necessary complement of navigation and search. We can think of a global list of top items as a list of personalized recommendations for an artificial "average user" (or a special case of personalized recommendations, shown to users with no ratings), and the conclusions from the following discussion are likely to hold for calculating personalized recommendations, individual for every user.

Throughout the work I assume that minimizing RMSE of rating prediction is a good intermediate task and allows, after proper adaptations, to calculate good quality lists of recommendations. This section will justify to some degree this assumption.

The simply stated question "Which movie is the best?" needs to be made more precise. The answer differs for different users, and personalized recommendations are an attempt to answer which items are the best from a perspective of a chosen user. In this section considered is the case of using simple models containing only the global effects: global mean, movie bias and user bias. This limits the scope of the analysis to construing non-personalized, global lists of top items. I will focus on the problem, how to generate a list of top $10 - 20$ items, best on average (in some sense) for all users. Rankings obtained with different methods will be compared visually, and we will think how to create a good ranking evaluation measure (we have already examined to some extent the problem of evaluation in section 3.4).

Consider first a single-element list, i.e. how to choose one movie that is the best on average. Attempts to precise this task lead to several questions: what does it mean, that a movie is the best? Is it the movie for which our most accurate model predicts the largest probability of rating 5 or the lowest probability of rating 1 (for an average user)? Is it the movie, for which our model predicts the largest chance to have the largest average rating, in a hypothetical situation where all users watched and rated all movies? How important is a chance for high average rating, and how important is the risk of low average rating?

What does it mean, that a movie is the best on average or the best for an average user? Are all users equally important? Do they have the same weight in the averaging process? When one movie is better than another movie?

Let's simplify the task even more. Let's assume, that we consider one user, and decide between two movies for him. Suppose for a moment that ratings are continuous. Let's assume, that the rating of one movie is drawn from a known distribution $N(m_1, s_1^2)$, and the rating of the second from $N(m_2, s_2^2)$. We can use here the framework of decision theory. The decision is to choose one of the movies, and each choice is evaluated by a utility function, that tells how the user values every rating distribution $N(m, s^2)$. How to choose the right utility function for the given user? If one distribution has a larger mean than the second one, but also has a much larger variance than the second distribution, should the first distribution be preferred? Clearly, a proper utility function will differ for different users. Different users can allow a different level of uncertainty about the predicted item rating. One user may want to have large precision, that is be sure that one the list of top items there will be few mistakes, items that will turn out to be weak (e.g. have low average, when they will be rated by more people). Another user may expect large recall, that is if there is a good movie somewhere, it should not be skipped from a list of top items, even if its predicted rating is uncertain because, for example, it was computed based on few gathered user ratings (these are different definitions of precision and recall than the usual definitions for a binary indicator of relevance). Similar dilemmas were mentioned in section 3.1, in the discussion of balancing a recommender system between recommending popular, proved content and long tail content, and in section 3.4 "Evaluation". The user's decision is psychological, and to make a reasonable choice of utility function, one could conduct surveys on a group of users, and then, with machine learning techniques, extrapolate results to all users. I do not have this kind of additional data, so my analysis has to be simplified and based on the data I have.

The analyzed situation complicates when comparing a large set of candidate movies. If I want to perform a test if a given movie is the best in a set, I would compare a Gaussian distribution with a maximum of multiple Gaussian distributions, which has the form of an extreme value distribution (it should not make much difference, if we treat this maximum as a fixed threshold, ignoring its uncertainty). Such test is performed for all movies, creating a situation of multiple comparisons, and hence largely amplifying the influence of uncertainty of predictions. Considering whole lists instead of only one top movie complicates the problem further. Finally, a large issue in determining top movies is considering the missing data distribution, that means, considering what causes that a movie is watched and rated – because this can significantly influence the observed average rating. Various measures evaluating rankings have been proposed that can be used to evaluate lists of top items (see also sections 3.1 and 3.4), such as mean average precision or NDCG, and one can search for lists optimizing the chosen measure. Those measures have a disadvantage, that optimizing them directly has large computational complexity, and a larger disadvantage, that they use only the training data, ignoring the missing data mechanism. In real-world tasks usually we can find ways to bypass any computational complexity issues. It turns out that in practice simplified approaches work well enough: a way to evaluate quality (score) of a single item is chosen, and items are sorted according to this score. Because in this work I describe methods that minimize RMSE, I am especially interested in indirect approaches that base on rating prediction (as we shall see, besides the expected ratings of items, in order to obtain good quality lists of top-k items also assessing uncertainty of predictions will be needed). Yet another issue to consider is what to do with similar items on a top-list – whether they are desirable or undesirable.

Let's look first at the disadvantages of simplest methods that calculate top rankings. One of the easiest ways to calculate a top ranking is to sort movies simply according to

Table 8: Ranking by arithmetic mean: top 15 movies.

	Title	Avg. of ratings (score)	Count of ratings	Freq. rank
1.	Lord of the Rings: The Return of the King: Extended Edition	4.723	72600	306.
2.	Lord of the Rings: The Fellowship of the Ring: Ext. Ed.	4.716	72274	303.
3.	Lord of the Rings: The Two Towers: Extended Edition	4.702	73630	295.
4.	Lost: Season 1	4.678	5758	2522.
5.	Battlestar Galactica: Season 1	4.669	1436	5603.
6.	Fullmetal Alchemist	4.597	1565	5774.
7.	The Shawshank Redemption: Special Edition	4.593	137812	51.
8.	Ghost in the Shell: Stand Alone Complex: 2nd Gig	4.586	174	12618.
9.	Trailer Park Boys: Season 4	4.583	24	17761.
10.	The Simpsons: Season 6	4.577	7967	2308.
11.	Tenchi Muyo! Ryo Ohki	4.576	85	17183.
12.	Veronica Mars: Season 1	4.570	1049	6490.
13.	Lord of the Rings: The Return of the King: EE: Bonus Mat.	4.563	119	15627.
14.	Arrested Development: Season 2	4.559	5763	2674.
15.	Trailer Park Boys: Season 3	4.559	68	17522.
	average of top 15	4.617	25355	7136

the arithmetic mean of item ratings in the training set: $Score_j = \frac{1}{|I_j|} \sum_{i \in I_j} r_{ij}$. Table 8 shows top 15 items with this criterion for the Netflix Prize data. We see that sorting by average rating is not a satisfactory solution. The DVD "Trailer Park Boys: Season 4" with only 24 ratings is in 9th place in the ranking.

Models with movie bias allow for another global ordering of movies. For example, in the described earlier Bayesian model with user and movie biases $\mu + c_i + d_j$, the posterior distribution of a per-movie variable d_j is Gaussian with expected value Ed_j and variance $Var\ d_j$. Let's see what happens if we sort movies according to Ed_j. In comparison to sorting by average rating, Ed_j estimates are regularized (here shrinked towards zero), and account for user bias. The resulting ranking is shown in table 9. The last column lists tripled standard deviation as a measure of uncertainty. If d_j is drawn from $N(Ed_j, Var\ d_j)$ distribution, d_j lies in the range $Ed_j \pm 3\sqrt{Var\ d_j}$ with probability larger than 99.7%.

Table 9: Ranking by expected movie bias Ed_j: top 15 movies.

	Title	Avg. of ratings	Cnt. of ratings	Freq. rank	$\mu + Ed_j$ (score)	$3\sqrt{Var\ d_j}$
1.	Battlestar Galactica: Season 1	4.669	1436	5603.	4.690	0.073
2.	Lost: Season 1	4.678	5758	2522.	4.690	0.036
3.	Lord of the Rings: The Return of the King: EE	4.723	72600	306.	4.673	0.010
4.	Lord of the Rings: The Fellowship of the Ring: EE	4.716	72274	303.	4.668	0.010
5.	Lord of the Rings: The Two Towers: EE	4.702	73630	295.	4.654	0.010
6.	Veronica Mars: Season 1	4.570	1049	6490.	4.633	0.085
7.	Arrested Development: Season 2	4.559	5763	2674.	4.629	0.036
8.	The Sopranos: Season 5	4.532	20196	1200.	4.576	0.019
9.	The Simpsons: Season 6	4.577	7967	2308.	4.567	0.031
10.	The Shawshank Redemption: Special Edition	4.593	137812	51.	4.564	0.007
11.	Trailer Park Boys: Season 3	4.559	68	17522.	4.553	0.326
12.	Band of Brothers	4.512	36850	694.	4.544	0.014
13.	The West Wing: Season 3	4.469	6433	2667.	4.543	0.034
14.	House, M.D.: Season 1	4.497	1313	5821.	4.540	0.076
15.	Anne of Green Gables: The Sequel	4.438	649	8190.	4.533	0.108
	average of top 15	4.586	29587	3776	4.604	0.058

We see that sorting by expected d_j has a similar disadvantage as sorting by average rating. "Trailer Park Boys: Season 3" having only 68 votes is in 11th place. It is a good

TV series with average rating 9.3/10 on IMDb (update: now it is 8.6/10 in 2012), but with so few ratings in the Netflix Prize dataset it should not be in the top 15 here.

A better solution is to take into account the posterior standard deviation of the learned parameter d_j. We can suspect, that a good scoring criterion used for sorting should reward a high mean rating, but should penalize high variance of prediction, because high variance increases the risk of making a bad recommendation for a user. We can try a following score function: $score_j = Ed_j - 3\sqrt{Var\ d_j}$ (see section 3.4 "Evaluation" for some justification of this formula). Then, if d_j is actually drawn from $N(Ed_j, Var\ d_j)$, then $d_j > score_j$ with probability larger than 99.8%. Table 10 shows the ranking sorted by $score_j$. The resulting ranking has visibly better precision than two previous ones. The DVD "Veronica Mars: Season 1" has the least ratings, 1049.

Table 10: Ranking by expected movie bias corrected by triple standard deviation $Ed_j - 3\sqrt{Var\ d_j}$: top 15 movies.

	Title	Avg. of ratings	Cnt. of ratings	Freq. rank	$\mu + Ed_j$	$3\sqrt{Var\ d_j}$	$\mu + Ed_j$ $-3\sqrt{Var\ d_j}$ (score)
1.	LotR: The Return of the King: EE	4.723	72600	306.	4.673	0.010	4.663
2.	LotR: The Fellowship of the Ring: EE	4.716	72274	303.	4.668	0.010	4.658
3.	Lost: Season 1	4.678	5758	2522.	4.690	0.036	4.654
4.	LotR: The Two Towers: EE	4.702	73630	295.	4.654	0.010	4.643
5.	Battlestar Galactica: Season 1	4.669	1436	5603.	4.690	0.073	4.618
6.	Arrested Development: Season 2	4.559	5763	2674.	4.629	0.036	4.593
7.	The Sopranos: Season 5	4.532	20196	1200.	4.576	0.019	4.557
8.	The Shawshank Redemption: SE	4.593	137812	51.	4.564	0.007	4.557
9.	Veronica Mars: Season 1	4.570	1049	6490.	4.633	0.085	4.548
10.	The Simpsons: Season 6	4.577	7967	2308.	4.567	0.031	4.536
11.	Band of Brothers	4.512	36850	694.	4.544	0.014	4.530
12.	The West Wing: Season 3	4.469	6433	2667.	4.543	0.034	4.509
13.	The Simpsons: Season 5	4.542	17069	1423.	4.524	0.021	4.503
14.	The Godfather	4.504	105707	130.	4.507	0.008	4.498
15.	Seinfeld: Season 3	4.499	9084	2162.	4.527	0.029	4.498
	average of top 15	4.590	38242	1922	4.312	0.028	4.571

Preparing lists of top items is a task commonly encountered in practice. I will describe now a method used currently (2011 year) by IMDb (The Internet Movie Database) to calculate top 250 movies based on users' ratings. This method is used also on several other websites that gather ratings. The IMDb method scores items as follows (only ratings of regular voters are used):

$$score_j = \frac{N_j}{N_j + \lambda} \sum_{i, r_{ij} \in R_j} r_{ij} + \frac{\lambda}{N + \lambda}\ \mu = \mu + \frac{1}{N_j + \lambda} \sum_{i, r_{ij} \in R_j} (r_{ij} - \mu)$$

Movies with the number of ratings $N_j < \lambda$ are discarded and don't appear in the top-K list.

In the version of scoring used to calculate top 250 movies on IMDb, the parameter λ was set to 1500. The algorithm can be understood as assuming a common prior distribution for all movies, and MAP point estimation for every movie. A large value of λ corresponds to strong a-priori anticipation that a rating will be close to the global average μ (in other words, the variance of the prior distribution is small). The chosen strong prior makes difficult manipulating the ranking with shilling-type attacks, when multiple users agree to rate high or low a chosen item. I can speculate that λ in IMDb algorithm is artificially high to counteract both the uncertainty of MAP estimates (selecting movies to the top-K is a multiple comparisons task – we have multiple chances to make a mistake) and the

possibility of ratings being not independent (like in situations of shilling attacks, which effects are to prevent).

Table 11: Ranking by IMDb score: top 15 movies.

	Title	Avg. of ratings	Count of ratings	Freq. rank	IMDb score
1.	Lord of the Rings: The Return of the King: Ext. Ed.	4.723	72600	306.	4.646
2.	Lord of the Rings: The Fellowship of the Ring: Ext. Ed.	4.716	72274	303.	4.639
3.	Lord of the Rings: The Two Towers: Ext. Ed.	4.702	73630	295.	4.626
4.	The Shawshank Redemption: Special Edition	4.593	137812	51.	4.553
5.	Lord of the Rings: The Return of the King	4.546	133597	64.	4.505
6.	Star Wars: Episode V: The Empire Strikes Back	4.544	91187	192.	4.485
7.	Raiders of the Lost Ark	4.504	117456	93.	4.458
8.	The Godfather	4.504	105707	130.	4.454
9.	Star Wars: Episode IV: A New Hope	4.505	84480	232.	4.442
10.	Lord of the Rings: The Two Towers	4.461	150676	30.	4.425
11.	Schindlers List	4.458	100518	155.	4.405
12.	Star Wars: Episode VI: Return of the Jedi	4.461	88041	215.	4.401
13.	Lord of the Rings: The Fellowship of the Ring	4.434	147932	33.	4.398
14.	Finding Nemo (Widescreen)	4.415	139050	47.	4.377
15.	Band of Brothers	4.512	36850	694.	4.370
	average of top 15	4.539	103454	189	4.479

Table 11 shows the result of applying the IMDb sorting to the Netflix Prize data. We see, that to get on the top 15 list in the IMDb method of ranking, a movie must have not only large average rating, but also has to be very popular. The least frequently rated movie on top 15 list is "Band of Brothers" with 36,850 ratings.

Some machine learning methods (and in particular a large part of the collaborative filtering methods described in this work) can approximate the expected value, but do not give an estimate of variance, so it may be useful to have a way to calculate a good quality ranking of items only from the expected values, without having accurate estimates of variances. For a movie bias d_j, the standard deviation of the posterior distribution should be roughly inversely proportional to the square root of the number of observations (ratings given to movie j). I correct the ranking in the table 8 by subtracting $C * N_j^{-0.5}$ from the average ratings. For large C the top 15 list becomes very similar to the list made by IMDb method. Table 12 shows the ranking for $C = 40$:

The IMDb approach (table 11) corrects the average rating by an $O(1/N_j)$ term, where N_j is the number of ratings of movie j, and the method in table 12 corrects the average rating by an $O(1/\sqrt{N_j})$ term, a crude estimate of the difference between posterior standard deviations of items. The difference between these two ways of correcting the average seems large, but the resulting rankings in tables 11 and 12 are almost identical (only places 12th and 13th are swapped). Both approaches use scores that with very high probability bound the true unknown average (for the data distribution) from below. The precision (understood as a chance of the top 15 containing no mistakes – no items that a user will rate low) is higher than the earlier used in table 10 correction by tripled standard deviation, at the cost of skipping items with larger, but uncertain predicted means.

The structure of which movies were rated (the structure of missing data) influences the observed statistics, such as the observed average rating. If randomly selected users were forced to rate the given movie (or better, forced to watch and rate), the movie average would be much lower than the observed one (see [Mar08]). It makes a difference whether an item is exposed to a group of users who like it, or to a group of users who do not like it. The observed rating average depends on the website traffic patterns, the distribution channel (movie theater, TV, DVD or VOD), item recurrence (a single movie, sequel of a movie, TV series), and so on. For example, if someone rates all seasons of a TV series, he

Table 12: Ranking by arithmetic mean corrected by multiplicity ($C = 40$) of $1/\sqrt{N_j}$: top 15 movies.

	Title	Avg. of ratings $\overline{r}$	Count of ratings	Freq. rank	$CN_j^{-0.5}$	$\overline{r}-$ $-CN_j^{-0.5}$
1.	LotR: The Return of the King: EE	4.723	72600	306.	0.148	4.574
2.	LotR: The Fellowship of the Ring: EE	4.716	72274	303.	0.149	4.568
3.	LotR: The Two Towers: EE	4.702	73630	295.	0.147	4.554
4.	The Shawshank Redemption: Special Ed.	4.593	137812	51.	0.108	4.485
5.	LotR: The Return of the King	4.546	133597	64.	0.109	4.436
6.	Star Wars: Ep. V: The Empire Strikes Back	4.544	91187	192.	0.132	4.412
7.	Raiders of the Lost Ark	4.504	117456	93.	0.117	4.387
8.	The Godfather	4.504	105707	130.	0.123	4.381
9.	Star Wars: Ep. IV: A New Hope	4.505	84480	232.	0.138	4.367
10.	LotR: The Two Towers	4.461	150676	30.	0.103	4.357
11.	Schindlers List	4.458	100518	155.	0.126	4.332
12.	LotR: The Fellowship of the Ring	4.434	147932	33.	0.104	4.330
13.	Star Wars: Ep. VI: Return of the Jedi	4.461	88041	215.	0.135	4.326
14.	Finding Nemo (Widescreen)	4.415	139050	47.	0.107	4.308
15.	Band of Brothers	4.512	36850	694.	0.208	4.304
	average of top 15	4.539	103454	189	0.130	4.408

is likely a fan of the series – if someone saw the first season and did not like it, it is unlikely that he saw and rated the remaining ones, and in effect, ratings for TV series are usually higher than movie ratings (this effect is visible, for example, in the top few thousands most frequently rated movies and TV series in IMDb). A similar effect can be spotted in movie sequels, for example, it may explain why "The Bourne Ultimatum" (2007) the third part of the Bourne trilogy, has highest average rating on IMDb of the three movies. We see that there are justified doubts about accuracy of top-K lists without modelling in any way the structure of missing data. To create a more accurate list of top-K items, we should consider a problem such as: what would be the average rating, if all users watched and rated the given movie. A few models described later in this work to some degree model the structure of missing data, for example, Conditional RBM [Sal07a] or SVD++ [Bel07e], but those methods were optimized to predict ratings from the training data distribution.

Evaluating and comparing whole top-lists can be useful, and from that perspective we see that accurate evaluation of top items (and their ordering) is more important than accurate evaluation of items with likely low rating. In [Kor08] the following ranking-based method of assessing quality of top-K recommendations was proposed: for each of the $384,573$ five-star ratings from the Netflix probe set a prediction is calculated, and compared with the prediction for 1000 random movies to be rated by the same user. The $1000 + 1$ items are sorted by predicted rating, and the resulting rank (percentile) in the set of the known relevant item is stored. After gathering the $384,573$ predicted values for all relevant observations, we get a distribution of percentiles, which can be compared for different algorithms. I used the evaluation criterion [Kor08] to compare rankings for the model of two biases (the model used to produce the ranking in table 10), for different choices of α, when evaluating items by $Ed_j - \alpha\sqrt{Var\ d_j}$. The top-1.5% accuracy (top-15 of 1001) was 14% for $\alpha = 0$ (ranking in table 9), 16% for $\alpha = 3$ (ranking in table 10), and the optimal α was about 400, with 33% top-1.5% accuracy, that is over 2.5 times larger probability of entering top-1.5%, than when sorting by expected rating. The optimal coefficient is large ($\alpha \approx 400$) likely because selecting top-K set among N items with uncertain scores (described by posterior rating distributions) is a situation of multiple comparisons (many items have a chance to exceed the score threshold needed to enter the top-K list). Similarly large differences in evaluating accuracy of rankings should appear when applying variance-based corrections of different size to more complex algorithms described later

in the work. This was the only experiment I performed with evaluating rankings. In the rest of the work I focused on the well defined task of predicting the expected rating, and assumed that there exist efficient ways to adapt the resulting algorithms to calculate good quality personalized recommendations.

The above method shares disadvantages of other commonly used ranking-based evaluation measures, such as MAP (mean average precision - average precision at the rank of each relevant item), ATOP (area under top-k recall curve) [Ste10b], NDCG (Normalized Discounted Cumulative Gain), or relaxed versions of these kinds of measures [Yue07]. Those typically used measures ignore the missing data structure, and also, direct optimization of ranking-based measures may have too large computational complexity [Wei07] (useful can be simplifications, such as based on neural networks [Bur05, Ren09, Jah11b]). Attempts to modify the ranking-based measures to account for missing data are not completely satisfying so far, for example, the task in Track 2 of the KDD Cup 2011 [Dro11] is to distinguish high ratings from specially sampled missing data, a criterion that, in my opinion, puts too much weight on predicting the missing data (which does not have to be closely connected to user likes and dislikes), while ignoring the valuable information contained in low ratings.

The amount of similarity between items on top-K lists should also be evaluated. Filling a short list of top-K items (or a list of personalized recommendations) with very similar items is undesirable, and the scoring function should penalize presence of similar items. We could design a criterion evaluating lists that penalizes dependencies between prediction errors, for example by rewarding high probability that at least one item from the proposed list will enter the true top-K list. Optimizing such joint criterion may have too large complexity, so a better idea may be deciding on a simpler heuristic solution: removing an item from the calculated top-list, if it is similar to another item higher on the list. Another heuristic solution (mentioned in [Pat07]) is to calculate clustering of items, and allowing on a top-K list only items from different clusters. A related requirement is that some items should not be recommended before another items are rated by the user, for example a sequel typically should not be recommender before the first movie, and subsequent seasons of a TV series should not be recommended before rating (watching) the previous seasons.

Another issue in practical applications is that a system of personalized recommendations, and even a simple top item list, are systems with loopback: the displayed lists of items depend on users' ratings, but the lists are part of website navigation, and which items are rated by a user depends on which items are displayed. Presence of the loopback construction causes a risk that if we too much concentrate on precision of recommendations, not allowing for discovery of very good items, then new items will not have a chance to get many ratings. In recommender systems based on implicit feedback, like clickstream data, there is yet another risk, that the recommendation lists will become polluted by items, that became popular accidentally, because of being exposed in website navigation or because they are inaccurately described, and their popularity was further amplified by the recommender system (or by reaching a list of top items). In clickstream-only based systems a user does not have an easy possibility to indicate, that he does not like a recommended item.

In summary, the subject of learning to rank is broad and, looking at applications, connected to human-computer interaction, interface design, psychometrics, advertising research, search, and other domains. The main focus of this work is rating prediction, so I stop on the above preliminary analysis of the problem of ranking items. To perform a more complete analysis more data is needed, such as proper surveys or elicitations, that would tell more about the user's perspective.

The most important conclusion from the simplified analysis is that to calculate a good

quality list of top items (or a list of recommendations) it is not enough to estimate the expected rating of items. It is necessary to take into account the uncertainty of prediction, which decreases with increasing number of collected ratings for an item. The proposed correction is in the form of decreasing the expected rating by padding dependent on the predicted uncertainty of a rating. Two types of correction worked well: by a factor of $O(1/\sqrt{N_j})$ and by a factor of $O(1/N_j)$, where N_j is the number of ratings of movie j. Which sorting criterion is the right one, this depends on how we evaluate errors in top-K lists (how we define our loss function). I advocate in this work using a heuristic – sorting by the predicted expected rating corrected by subtracting $C/\sqrt{N_j + \lambda}$ (the form of correction was to some extent justified in section 3.4 "Evaluation"). The amount of correction should be tuned to determine the right balance between recommending popular, trustworthy content, and allowing for content discovery among new items or niche items from the "long tail" (items with few ratings). The amount of correction should be user-specific, because users can have different expectations about taking risks, but for simplicity we can use one global constant C. Because selecting a few items from a set is a multiple comparisons situation, the amount of correction C grows with the number of compared items – a different sorting criterion is needed when we calculate a top-10 list among 50 items, among 1000 items, or among a million items.

Other issues to consider are understanding and correcting for the missing data structure, which can contain artifacts caused by interaction of recommendations with website navigation, and amplified by loopback effects. Which groups of items are exposed to which groups of users, this influences the observed average rating. Also, a solution ensuring diversity of lists can be needed.

Comparing to selecting the global top-K by average rating, in my judgement the most important corrections are, in order of importance:

1. corrections for uncertainty of predictions (posterior variance),
2. corrections for missing data structure (can be more important in other datasets, depending on how the ratings are gathered).
3. regularized estimates (choice of priors in the Bayesian approach; regularization becomes more important in the algorithms for personalized recommendations).

4.2.5 One-feature regularized SVD with biases

Concluding the part describing simple models, in this section I examine extensively a simplified method of regularized singular value decomposition with biases. The simplification lies in that the model contains, besides biases, only one feature, with one parameter for each user to model the user preference, and one parameter for each movie.

The one-feature model allows to reflect on foundations of full, multi-feature regularized SVD models, which proved very effective for the task of rating prediction. The performed review of modelling choices, such as selecting the right prior distributions for hidden variables, and scrutinizing the idea of multiplying hidden variables, may also be useful in a broader context, in applications other than collaborative filtering. For example, similar structures with multiplication of hidden variables are typically a component of methods in the general family of neural networks.

In the Netflix Prize task the most effective methods were based on multidimensional regularized SVD, called also matrix factorization. In further parts of the work (section 4.3) I describe different variants of regularized SVD, with the number of features between $K = 30$ and $K = 200$ in my implementations. The multidimensional model has the form:

$$\hat{r}_{ij} = \mu + c_i + d_j + \sum_k u_{ik} v_{jk} = \mu + c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j$$

The parameter v_{jk} denotes the level of the k-th feature of movie j, and can be understood as an automatically learned movie genre. The parameter u_{ik} denotes the level of preference of user i for k-th movie genre. Such automatically learned movie genres do not necessarily have lot in common with the genres named by human experts, like action movie, comedy, horror. It can be a combination of many named genres. It may also express some common traits of movies, that have not yet been named.

The biases c_i and d_j are a variant of global effects [Fun06, Bel07b, Pot08, Tos08b]. Usually a little better accuracy was obtained by treating biases c_i and d_j as a component of the model [Pat07, Tak07a, Tak07b, Tak09a, Bel08] than learning biases separately from the main model, in a phase of preprocessing by removing global effects.

This section examines a simplified version of the above model, that contains only one feature (hidden genre):

$$\hat{r}_{ij} = \mu + c_i + d_j + u_i v_j$$

I will compare from the perspective of predictive accuracy different possible ways of learning parameters. I will reflect on what is the best form of a prior distribution for parameters. At the end I will summarize experiments with nonparametric modelling of a two-parameter function connecting user preferences with a movie feature. Its goal was to verify, if multiplying $u_i v_j$ in SVD is the right choice for expressing the hidden relationship.

Note that the general problem of calculating approximate SVD with missing data is NP-hard, even for one-dimensional approximation [Gil10], but in practice, for the Netflix data, regularized SVD methods with a right choice of optimization way seem to converge to solutions close to global minima.

Table 13 shows a comparison of accuracy for different methods of learning the parameters c, d, u, v. In these experiments the entire probe set was used as the test set. Let's describe the methods listed in the table.

The first listed method, "SVD gradient descent, MAP biases once", is regularized SVD (with one feature) based on RMSE cost function minimization with regularization. The method was proposed in [Fun06], with biases learned as preprocessing before learning u_i, v_j parameters. The regularized cost function to minimize is following:

$$l = \sum_{ij \in Tr} (r_{ij} - \mu - c_i - d_j - u_i v_j)^2 + N_i \frac{\lambda_u}{2} u_i^2 + N_j \frac{\lambda_v}{2} v_j^2 + \frac{\lambda_c}{2} c_i^2 + \frac{\lambda_d}{2} d_j^2.$$

The parameters u_i,v_j are learned by following the negated first derivative of the cost function, resulting in the following updates:

$$res_{ij} = r_{ij} - \mu - c_i - d_j - u_i v_j$$

$$u_i \mathrel{+}= lrate * (res_{ij} * v_j - \lambda_u * u_i)$$

$$v_j \mathrel{+}= lrate * (res_{ij} * u_i - \lambda_v * v_i)$$

The constant parameters were fixed to values proposed in [Fun06]: $lrate = 0.001$, $\lambda_u = \lambda_v = 0.02$. Biases were optimized once as a preprocessing, with constant regularization $\lambda_c = \lambda_d = 5.0$.

Accuracy is improved by treating biases c_i, d_j as part of the model ("SVD gradient descent, MAP biases simul."), and learning them simultaneously with u_i, v_j [Pat07, Tak07a, Tak07b] ([Pat07] used a special regularization of biases $(N_i + N_j)\lambda_{cd}(c_i + d_j)^2$, chosen experimentally among several alternatives).

The next group of methods are approximations of Bayesian inference of parameters, which are treated now as random variables. A prior distribution of parameters $\theta = (u_i, v_j, c_i, d_j)$ is assumed and we want to calculate the posterior distributions on

Table 13: One-feature SVD with biases. Comparison of experimental results.

Method	Parameters	Iterations	RMSE
SVD grad. descent, MAP biases once	$lrate = 0.001, \lambda_{uv} = 0.02, \lambda_{cd} = 5.0$	100	0.9625
SVD grad. descent, MAP biases simul.	$lrate = 0.001, \lambda_{uv} = 0.02, \lambda_{cd} = 5.0$	100	0.9531
SVD Bayes., Gibbs, avg. expectation	$\tau_v^2 = 1$	10 + 40	0.9520
SVD Bayes., Gibbs, avg. of samples	$\tau_v^2 = 1$	10 + 40	0.9524
SVD Variational Bayesian	$\tau_v^2 = 1$	20	0.9510
SVD Variational Bayesian	$\tau_u^2 = 1$	20	0.9510
SVD MAP, opt.	$\lambda_{cd} = 5, \lambda_u = (4.0, 0.04), \lambda_v = (6.0, 0.011)$	10	0.9511
SVD gradient descent w/opt. param.	$lrate = 0.001, \lambda_{cd} = 5, \lambda_u = (4.0, 0.04), \lambda_v = (6.0, 0.011)$	100	0.9518
SVD gradient descent, reg. $\|u_i\|^3$, $\|v_j\|^3$	$lrate = 0.002, \lambda_{cd} = 5, \lambda_u = 1.5 * (4.0, 0.04), \lambda_v = 1.5 * (6.0, 0.011)$	100	0.9515
SVD MAP, regularization $\|u_i\|^3$, $\|v_j\|^3$	$\lambda_{cd} = 5, \lambda_u = 1.5 * (4.0, 0.04), \lambda_v = 1.5 * (6.0, 0.011)$	10	0.9512
SVD Bayesian, nonparam. prior		20 + 50 + 50	0.9578
Nonparametric rel., avg. samples		100 + 100	0.9498

these parameters according to Bayes' rule: $p(\theta|D) \propto p(D|\theta)p(\theta)$, where D are the available data (observations), and θ is a vector of all model parameters. Calculations directly by Bayes' rule have here too large computational complexity to be applicable in practice, and it is necessary to use approximations. Two ways of approximation were used here for parameter inference in the one-feature SVD model: the first is MCMC with Gibbs sampling [Har07, Tom07, Sal08], and the second is Variational Bayes [Lim07, Rai07].

In the probabilistic model used, prior distributions of each parameter are independent Gaussians, differently parameterized for each group of parameters c, d, u, v. We can say that the model has a property of exchangeability of users (users are indistinguishable a-priori, before seeing the data) and exchangeability of movies.

$$c_i \sim N(\overline{c}, \tau_c^2) \qquad d_j \sim N(\overline{d}, \tau_d^2)$$
$$u_i \sim N(\overline{u}, \tau_u^2) \qquad v_j \sim N(\overline{v}, \tau_v^2)$$
$$r_{ij} \sim N(c_i + d_j + u_i v_j, \sigma^2)$$

The hyperparameters $\tau_c^2, \tau_d^2, \tau_u^2, \tau_v^2$ are point estimated by maximum likelihood in an empirical Bayes approach, where c, d, u, v are either sampled with MCMC, or their posterior distributions from VB approximation are used. The term $u_i v_j$ in the model implies redundancy between the parameters τ_u and τ_v, so I will fix either $\tau_u = 1$ or $\tau_v = 1$, and estimate only the second parameter. In a similar way the parameter σ^2 is learned by maximum likelihood in an empirical Bayes approach.

The posterior distribution $p(c, d, u, v|D)$ will be approximated by MCMC or Variational Bayes. First I describe the MCMC method with the Gibbs sampler (denoted in table 13 as "SVD Bayesian, Gibbs"), which is simpler than VB, but needs many more iterations. The method used largely resembles the John Tomfohr's model [Tom07]. In the Gibbs sampling method samples are generated from the joint posterior distribution $p(c_i, d_j, u_i, v_j|D)$ by drawing alternately from $p(c_i|D, d_j, u_i, v_j)$, $p(d_j|D, c_i, u_i, v_j)$, $p(u_i|D, c_i, d_j, v_j)$ and $p(v_j|D, c_i, d_j, u_i)$. Samples obtained in subsequent iterations are dependent on samples in previous iterations.

The conditional posterior distributions of c_i, d_j are calculated similarly to the previously described method with biases (section 4.2.3):

$$p(c_i|D, d_j, u_i, v_j) \propto \exp\Big(-\frac{(c_i - Ec_i)^2}{2\ Var\ c_i}\Big) \qquad p(d_i|D, c_i, u_i, v_j) \propto \exp\Big(-\frac{(d_i - Ed_i)^2}{2\ Var\ d_i}\Big)$$

$$Var\ c_i = \frac{1}{|J_i|/\sigma^2 + 1/\tau_c^2} \qquad Var\ d_j = \frac{1}{|I_j|/\sigma^2 + 1/\tau_d^2}$$

$$Ec_i = \Big(\frac{\sum_{j\in J_i}(r_{ij} - \mu - d_j - u_i v_j)}{\sigma^2} + \frac{\overline{c}}{\tau_c^2}\Big) * Var\ c_i$$

$$Ed_j = \Big(\frac{\sum_{i\in I_j}(r_{ij} - \mu - c_i - u_i v_j)}{\sigma^2} + \frac{\overline{d}}{\tau_d^2}\Big) * Var\ d_j$$

The parameters $\overline{c}, \overline{d}, \tau_c^2, \tau_d^2$ are estimated, similarly as earlier in section 4.2.3, by maximum likelihood from posterior samples of c_i, d_j (the method can be called Monte Carlo Empirical Bayes):

$$\overline{c} = \frac{1}{N}\sum_{i=1}^{N} c_i \quad \tau_c^2 = \frac{1}{N}\sum_{i=1}^{N}(c_i - \overline{c})^2 \qquad \overline{d} = \frac{1}{M}\sum_{j=1}^{M} d_j \quad \tau_d^2 = \frac{1}{M}\sum_{j=1}^{M}(d_j - \overline{d})^2$$

The conditional posterior distribution for u_i in the Gibbs sampling has the form:

$$p(u_i|D, v_j, c_i, d_j) \propto p(D|u_i)p(u_i) = f(u_i) = \exp\Big(-\frac{\sum_{j\in J_i}(r_{ij} - \mu - c_i - d_j - u_i v_j)^2}{2\sigma^2} - \frac{(u_i - \overline{u})^2}{2\tau_u^2}\Big)$$

$$= \exp\Big(-C + u_i * \Big(\frac{\sum_{j\in J_i}(r_{ij} - \mu - c_i - d_j) * v_j}{\sigma^2} + \frac{\overline{u}}{\tau_u^2}\Big) - \frac{1}{2}u_i^2 * \Big(\frac{\sum_{j\in J_i} v_j^2}{\sigma^2} + \frac{1}{\tau_u^2}\Big)\Big)$$

We see that, with all remaining parameters fixed, the posterior distribution of u_i is Gaussian:

$$p(u_i|D, v_j, c_i, d_j) \propto \exp\Big(-\frac{(u_i - Eu_i)^2}{2\ Var\ u_i}\Big)$$

$$Var\ u_i = \frac{1}{(\sum_{j\in J_i} v_j^2)/\sigma^2 + 1/\tau_u^2}$$

$$Eu_i = \Big(\frac{\sum_{j\in J_i}(r_{ij} - \mu - c_i - d_j)}{\sigma^2} + \frac{\overline{u}}{\tau_u^2}\Big) * Var\ u_i \tag{20}$$

The hyperparameters $\overline{u}, \tau_u^2$ of the common prior distribution of u_i's are estimated, similarly to the hyperparameters of c and d, by maximum likelihood point estimation on a set of posterior samples of u_i's:

$$\overline{u} = \frac{1}{N}\sum_{i=1}^{N} u_i \qquad \tau_u^2 = \frac{1}{N}\sum_{i=1}^{N}(u_i - \overline{u})^2$$

The inference of the posterior distributions of v_j's is analogous:

$$Var\ v_j = \frac{1}{(\sum_{j\in I_j} u_i^2)/\sigma^2 + 1/\tau_d^2}$$

$$Ev_j = \Big(\frac{\sum_{i\in I_j}(r_{ij} - \mu - c_i - d_j) * u_i}{\sigma^2} + \frac{\overline{v}}{\tau_v^2}\Big) * Var\ v_j \tag{21}$$

The hyperparameters $\overline{v}, \tau_v^2$ are point estimated based on sampled v_j values:

$$\overline{v} = \frac{1}{M}\sum_{j=1}^{M} v_j \qquad \tau_v^2 = \frac{1}{M}\sum_{j=1}^{M}(v_j - \overline{v_j})^2$$

The parameter σ^2 is estimated using sampled c_i, d_j, u_i, v_j:

$$\sigma^2 = \frac{1}{|Tr|} \sum_{ij \in Tr} (r_{ij} - \mu - c_i - d_j - u_i v_j)^2$$

Alternating Gibbs sampling of c_i, d_j, u_i, v_j was run for 50 iterations, with fixed parameter $\tau_v = 1$. The first 10 iterations were skipped as a "burn-in" phase, and predictions were made in two ways. The first was averaging sampled $c_i + d_j + u_i v_j$ from the last 40 iterations, which gave $\text{RMSE}_{probe} = 0.9524$. The second method was predicting by average of expected outputs: $Ec_i + Ed_j + Eu_i Ev_j$ from the last 40 iterations, which gave $\text{RMSE}_{probe} = 0.9520$.

Variational Bayesian (VB) inference [Lim07, Rai07, Bis06] for the one-feature SVD model ("SVD Variational Bayesian" in table 13) results in similar formulas as MCMC. The VB method assumes independence of four groups of parameters c, d, u, v in the posterior distribution: $q(c)q(d)q(u)q(v)$. Using variational methods, a probability density function in a factorized form $q(c)q(d)q(u)q(v)$ is found [Bis06] that minimizes the Kullback-Leibler divergence from the joint distribution $p(c, d, u, v|D)$. Minimization leads to solutions in the form $\log q(u) = C + E_{c,d,v} \log p(c, d, u, v, D)$, and analogous expressions for $q(v), q(c), q(d)$. The resulting method is similar to MCMC, but, instead of sampling, the distributions of c, d, u, v are alternately updated. Making the same assumptions about Gaussian priors as in the MCMC method, the posterior distributions of c, d, u, v in VB are Gaussians $N(Ec_i, Var\ c_i)$, $N(Ed_j, Var\ d_j)$, $N(Eu_i, Var\ u_i)$, $N(Ev_j, Var\ v_j)$.

The parametrization of the posterior distributions of biases c_i and d_j is following:

$$Var\ c_i = \frac{1}{|J_i|/\sigma^2 + 1/\tau_c^2} \qquad Var\ d_j = \frac{1}{|I_j|/\sigma^2 + 1/\tau_d^2}$$

$$Ec_i = \Big(\frac{\sum_{j \in J_i} (r_{ij} - \mu - Ed_j - Eu_i Ev_j)}{\sigma^2} + \frac{\overline{c}}{\tau_c^2} \Big) * Var\ c_i$$

$$Ed_j = \Big(\frac{\sum_{i \in I_j} (r_{ij} - \mu - Ec_i - Eu_i Ev_j)}{\sigma^2} + \frac{\overline{d}}{\tau_d^2} \Big) * Var\ d_j$$

Point estimation of the hyperparameters $\overline{c}, \overline{d}, \tau_c^2, \tau_d^2$:

$$\overline{c} = \frac{1}{N} \sum_{i=1}^{N} Ec_i \qquad \tau_c^2 = \frac{1}{N} \sum_{i=1}^{N} ((Ec_i)^2 + Var\ c_i) - \overline{c}^2$$

$$\overline{d} = \frac{1}{M} \sum_{j=1}^{M} Ed_j \qquad \tau_d^2 = \frac{1}{M} \sum_{j=1}^{M} ((Ed_j)^2 + Var\ d_j) - \overline{d}^2$$

The parametrization of the posterior distributions of u_i and v_j:

$$Var\ u_i = \frac{1}{\sum_{j \in J_i} ((Ev_j)^2 + Var\ v_j)/\sigma^2 + 1/\tau_u^2} \tag{22}$$

$$Eu_i = \Big(\frac{\sum_{j \in J_i} (r_{ij} - \mu - Ec_i - Ed_j) * Ev_j}{\sigma^2} + \frac{\overline{u}}{\tau_u^2} \Big) * Var\ u_i \tag{23}$$

$$Var\ v_j = \frac{1}{\sum_{j \in I_j} ((Eu_i)^2 + Var\ u_i)/\sigma^2 + 1/\tau_d^2} \tag{24}$$

$$Ev_j = \Big(\frac{\sum_{i \in I_j} (r_{ij} - \mu - Ec_i - Ed_j) * Eu_i}{\sigma^2} + \frac{\overline{v}}{\tau_v^2} \Big) * Var v_j \tag{25}$$

Point estimation of the hyperparameters $\overline{u}, \overline{v}, \tau_u^2, \tau_v^2$:

$$\overline{u} = \frac{1}{N}\sum_{i=1}^{N} Eu_i \qquad \tau_u^2 = \frac{1}{N}\sum_{i=1}^{N}((Eu_i)^2 + Var\ u_i) - \overline{u}^2$$

$$\overline{v} = \frac{1}{M}\sum_{j=1}^{M} Ev_j \qquad \tau_v^2 = \frac{1}{M}\sum_{j=1}^{M}((Ev_j)^2 + Var\ v_j) - \overline{v}^2$$

Point estimation of the parameter σ^2:

$$\sigma^2 = \frac{1}{|Tr|}\Big(\sum_{ij\in Tr}(r_{ij} - \mu - Ec_i - Ed_j - Eu_iEv_j)^2 + Var\ c_i + Var\ d_j$$

$$+(Eu_i)^2\ Var\ v_j + (Ev_j)^2\ Var\ u_i + Var\ u_i\ Var\ v_j\Big)$$

Table 13 listed results of running two variants of VB method for 20 iterations. In the first method $\tau_u = 1$ was fixed, and in the second one $\tau_v = 1$ was fixed. Both methods gave identical $\text{RMSE}_{probe} = 0.9510$, the best among the tested here SVD methods with one feature.

One inconsistency in the VB method used is that the prior distributions for u_i and v_j, the posterior distributions for u_i and v_j, and the output distribution r_{ij} are all Gaussians. Investigating this observation led to experiments with nonparametric priors and and proposing a different choice of priors (see later parts of the section).

Another observation, after comparing the equations (23),(25) of VB with the equations (20),(21) of MCMC, was that the VB method underestimates the posterior variance u_i and v_j. Comparing (20) and (23), the sampled v_j^2's changed to $(Ev_j)^2 + Var\ v_j$, and $y_{ij}v_j$ changed to $y_{ij}Ev_j$, where $y_{ij} = r_{ij} - \mu - Ec_i - Ed_j$. The first change to $(Ev_j)^2 + Var\ v_j$ is a good approximation of the influence of the v_j^2 term, but the second change $y_{ij}v_j$ to $y_{ij}Ev_j$ causes decreasing the variance. Approximate corrections of the posterior variances by $Var\ u_i := Var\ u_i + (Var\ u_i)^2 \sum_{j\in J_i} y_{ij}\ Var\ v_j$ and analogous corrections of $Var\ v_j$ were too small to significantly influence the RMSE accuracy.

Variational Bayesian SVD equations explain the phenomenon initially noticed experimentally [Fun06], that in the cost function minimization approaches to SVD (called here also a neural network approach to SVD) better than constant regularization, identical for each user (each movie), is the amount of regularization increasing linearly with the number of user ratings (movie ratings). Constant regularization is a typical choice in machine learning, used in neural networks (called there weight decay), structural risk minimization, kernel methods and in other approaches. In the Netflix Prize task a more refined regularization term leads to avoiding overfitting to a large extent, and to a large accuracy improvement. In the formula (22) for the posterior variance of u_i visible is a term $\sum_{j\in J_i} Var\ v_i/\sigma^2$, that grows linearly with the number of ratings collected (and similarly in the formula (24) for the posterior distribution for v_j).

If we assume the hyperparameters $\overline{u} = \overline{v} = 0$, the VB solution can be approximated by alternating MAP (maximum a-posteriori) estimation with a specially selected linear regularization $\lambda^{(1)} + \lambda^{(2)}|I_j|$. Earlier were described two SVD methods with linear regularization (but without the constant terms, $\lambda_u^{(1)} = \lambda_v^{(1)} = 0$), listed in table 13 as "SVD gradient descent". The method "SVD MAP, opt." from table 13 uses nearly the same cost function, but, instead of optimizing it with gradient descent, the four groups of parameters c, d, u, v are alternately optimized by a jump to the marginal minimum. Regularization is constant for parameters c_i and d_j, but increases linearly with the number of user ratings for estimating u_i and with the number of movie ratings for estimating v_j.

$$y_{ij} = r_{ij} - (\mu + c_i + d_j + u_i v_j)$$

$$c_i := \frac{\sum_{j \in J_i}(y_{ij} + c_i)}{|J_i| + \lambda_c} \qquad d_j := \frac{\sum_{i \in I_j}(y_{ij} + d_j)}{|I_j| + \lambda_d}$$

$$u_i := \frac{\sum_{j \in J_i}(y_{ij} + u_i v_j) v_j}{\sum_{j \in J_i} v_j^2 + \lambda_v^{(1)} + \lambda_v^{(2)} |J_i|} \qquad v_j := \frac{\sum_{i \in I_j}(y_{ij} + u_i v_j) u_i}{\sum_{i \in I_j} u_i^2 + \lambda_u^{(1)} + \lambda_u^{(2)} |I_j|}$$

In this method there are six regularization parameters: λ_c, λ_d, $\lambda_u^{(1)}$, $\lambda_u^{(2)}$, $\lambda_v^{(1)}$, $\lambda_v^{(2)}$. Their values were set by minimization of RMSE on the test set, performed by Praxis [Bre71] optimizer, from Fortran Netlib library. The resulting RMSE_{probe} of the method is 0.9511. The parameters minimizing RMSE were $\lambda_{cd} = 5.0, \lambda_u = (4.0, 0.04), \lambda_v = (6.0, 0.011)$ (the parameters were rounded, and the rounding did not change the RMSE_{probe}).

The method "SVD gradient descent w/opt. param." optimizes exactly the same cost function as "SVD MAP, opt.", but with gradient descent (a first-order method) instead of alternating jumps to the minimum (a second order method).

The methods "SVD gradient descent, regularization $|u_i|^3$, $|v_j|^3$" and "SVD MAP, regularization $|u_i|^3$, $|v_j|^3$" use a special regularization suggested by the experiments with nonparametric priors.

The method denoted in table 13 as "SVD Bayesian, nonparam. prior" explores the idea of using nonparametric priors. Methods described so far in this section were more or less accurate approximations of the Bayesian approach in a model that assumes independent Gaussians as prior distributions on hidden parameters. Good practice when identifying or verifying a model is to use first a model with more parameters than necessary, and then try to use noticed regularities, patterns, dependencies, known distributions to simplify the model, decrease the number of parameters. One may wonder, if a Gaussian is the right choice for the prior distributions. I attempted to verify it by learning priors in a nonparametric form and inspecting the learned distributions, similarly as I did it for the model of biases in section 4.2.3. Again, I use here a prior in the form of Dirichlet process with zeroed concentration parameter. The prior distribution is initialized here to a set of uniformly sampled values from a 501-valued grid of equidistant points in the range $[-5, 5]$. The method is called nonparametric, because the grid density can be increased (each grid value is a new parameter) with increasing number of users (movies). The resulting algorithm is very similar to the nonparametric algorithm for biases described earlier in section 4.2.3:

1. Initialize $\mathbf{c}, \mathbf{d}, \mathbf{u}, \mathbf{v}$ to random values from the discrete set $[-5, -4.98, -4.96, -4.94, ..., 4.98, 5]$
2. for (iter in 1..100)
3. update_c($\mathbf{c}$)
4. update_d($\mathbf{d}$)
5. update_u($\mathbf{u}$)
6. update_v($\mathbf{v}$)
7. $\overline{\mathbf{c}}, \overline{\mathbf{d}}, \overline{\mathbf{u}}, \overline{\mathbf{v}}$ = mean of $\mathbf{c}, \mathbf{d}, \mathbf{u}, \mathbf{v}$ vectors from iterations >= 50
8. predict using $\overline{c}_i + \overline{d}_j + \overline{u}_i \overline{v}_j$

The functions update_u(), update_v(), update_c(), update_d() are similar to the update functions in the algorithm for biases in section 4.2.3. The function update_u() is following (the remaining ones being analogous):

```
1.  update_u(u):
3.    u^new = random_shuffle(u)
4.    for (i in 1..480189) // loop over users
5.      l1 = log_likelihood_u(u_i, R_i)
6.      l2 = log_likelihood_u(u_i^new, R_i)
7.      if (exp(l1 − l2) > random())
8.        u_i = u_i^new
```

The function log_likelihood_u(x, J_i) returns the logarithm of the likelihood of $u_i = x$, given the observed data, and assuming that all c_i, d_j, v_j parameters are fixed. I assume here, that the data comes from the distribution $N(\mu+c_i+d_j+u_iv_j, \sigma^2)$, hence log_likelihood_u(x, J_i) $= \frac{1}{2\sigma^2}\sum_{j\in J_i}(r_{ij}-\mu-c_i-d_j-xv_j)^2$. The function random() returns a sample from uniform distribution $[0, 1]$, and the function random_shuffle($\mathbf{x}$) returns a random permutation of the vector $\mathbf{x}$, each permutation with the same probability.

As in the algorithm in section 4.2.3, I assume $\sigma^2 = 0.5$. Changing of this parameter to a more accurate value has little influence on the resulting distributions.

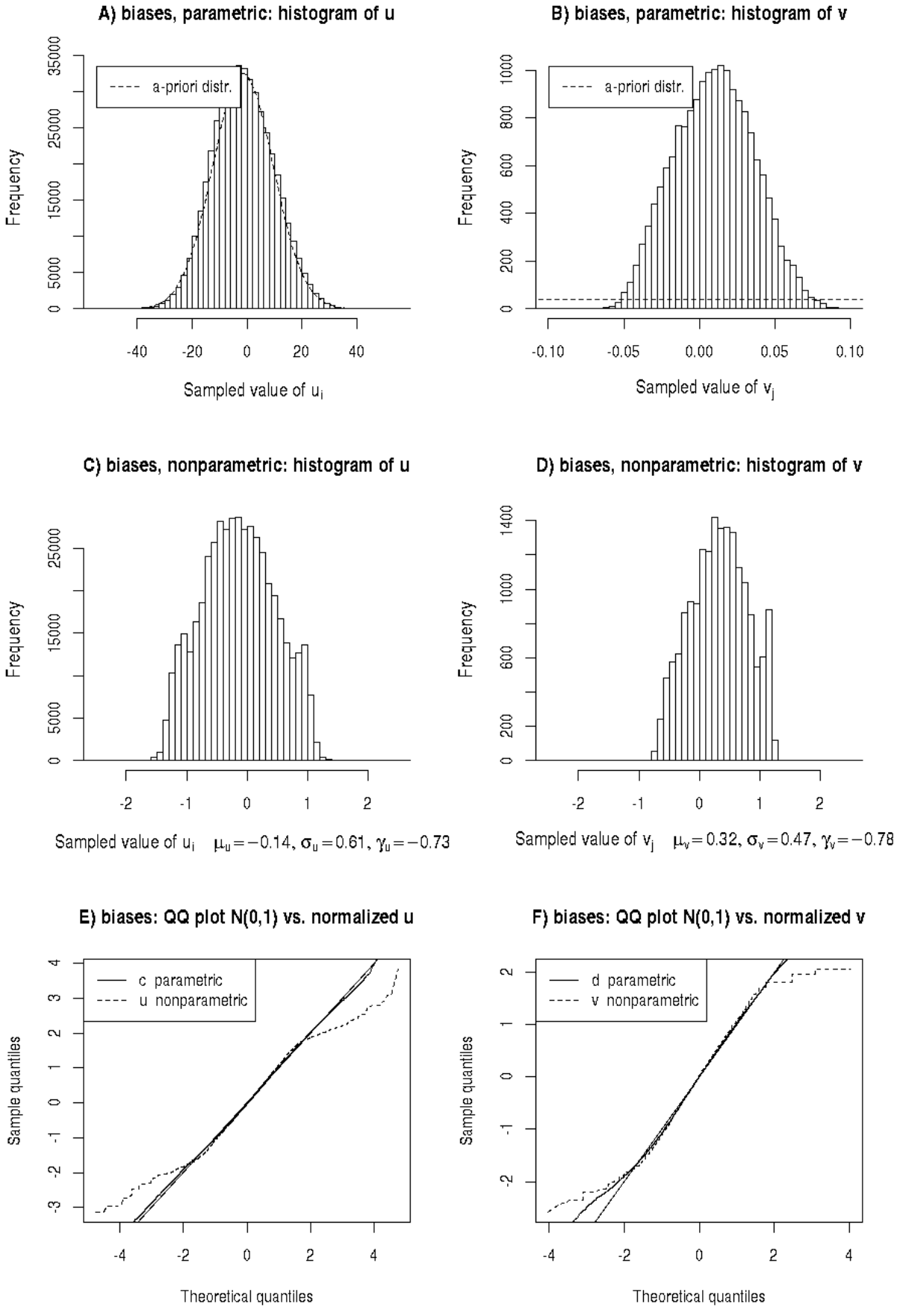
Let's visualize the posterior distributions, comparing the results of learning the parametric and nonparametric method. The resulting distributions of biases c, d have a similar shape to the learned in the "only biases" model (section 4.2.3), so to save space I will skip their visualization. Plots A, B in figure 27 visualize the posterior distributions of u and v for the parametric method (the one with Gibbs sampling). Plots C, D visualize the distributions for the nonparametric method (the set of samples from the posterior distributions becomes a new prior in the subsequent iteration). Plots E, F are quantile-quantile plots comparing the normalized (centralized and standardized) posterior distributions of u and v with the standard Gaussian $N(0, 1)$.

Summarizing the charts A and B in figure 27, for the parametric method the distribution of posterior samples of u_i roughly coincides with the prior distribution of u_i (estimated in the previous iteration of the algorithm). In turn, the posterior samples of v_j are concentrated in a narrow range, comparing to the v_j prior, which variance was fixed to $\tau_v^2 = 1$.

Interesting is the shape of the learned nonparametric priors on u and v observed on charts C, D, E, F. Noticeable are the kurtosis parameters $\gamma_u = -0.73$, and $\gamma_v = -0.78$ (the estimates may be inexact). In the parametric methods Gaussians were used as priors, for which the kurtosis is zero. An attempt of using different priors was using a generalized normal distribution: $p(x) \propto \exp(-C|x|^\alpha)$, which has negative kurtosis for powers $alpha > 2$. Initial experimental results showed a small accuracy improvement with $\alpha = 3$ (note that then the distribution is bimodal) over $\alpha = 2$ with gradient descent approach ("SVD gradient descent, regularization $|u_i|^3$, $|v_j|^3$" in table 13) and a small accuracy reduction when using alternating minimization ("SVD MAP, regularization $|u_i|^3$, $|v_j|^3$" in table 13). More experiments are needed to conclude whether a change of priors can significantly improve accuracy of the regularized SVD methods.

The learned nonparametric distributions for u and v with negative kurtosis, the form of the model $N(c_i+d_j+u_iv_j, \sigma^2)$ containing u_iv_j term, and the form of learned nonparametric priors for c_i and d_j (close to Gaussians) suggest that the term u_iv_j should be a-priori approximately Gaussian. This thought leads us to an interesting, simple question in the field of probability: which two symmetrical, iid variables X and Y after multiplying give a standard Gaussian variable $XY = Z \sim N(0, 1)$? As it often happens, simple to state questions about random variables can lead to complicated solutions. A question for which I do not know the answer is what is the pdf of the X and Y variables. What is known is that the moments are square roots of the standard Gaussian distribution (the odd moments are zero), so it is easy to calculate the kurtosis and compare with the observed values. The even moments of $N(0, 1)$ are $m_p = (p-1)!! = \Pi_i(2p-1)$. The even moments of the desired

Figure 27: Learned priors of u, v; in the parametric and nonparametric method



distribution of X and Y are hence $\sqrt{(p-1)!!}$. The kurtosis for the Gaussian distribution is $m_4/m_2^2 - 3 = 0$, and for the distribution of X and Y is $\sqrt{m_4}/m_2 - 3 = \sqrt{3} - 3 \approx -1.268$, so it is smaller than the observed one in our nonparametric method (with the remarks that our procedure of estimating the kurtosis may be inexact, and the output distribution is not precisely Gaussian).

Summarizing the experiments so far, better priors than Gaussian can be chosen in the probabilistic SVD model. Another thing to improve approximation of the MCMC method is to use a different method than VB with minimizing KL distance. Minimizing KL distance with Gaussian priors leads to Gaussian posteriors and hence approximating ratings with sums of spike-shaped distributions – normal product distributions. VB SVD with minimizing Hellinger distance could be tried out (but here calculations complicate).

To gain further insights into the SVD methods used here, I looked at synthetic data in the form $u_i v_j + \epsilon_{ij}$, where u_i and v_j are once drawn from $N(0,1)$, and each error ϵ_{ij} is drawn also from $N(0,1)$. A noticed problem is that the influence of priors on u and v (here $N(0,1)$) on the obtained posteriors is small. In 10 different draws of the data, and re-running a regularized SVD method I obtained the following estimates of variance $\hat{\tau}_u^2$: (0.11, 0.98, 0.01, 0.77, 0.42, 1.34, 0.22, 1.64, 0.55, 0.43). In multi-feature SVD appears another problem – often appearing multicollinearity (once per several draws of data) in one of the columns or rows, causing large increase of RMSE. We can speak therefore of bad conditioning of the task of recovering u and v parameters, when each u_i and v_j are drawn from Gaussians, and the data matrix is drawn from $N(u_i v_j, \sigma^2)$.

The last method listed in the table 13 is "nonparametric relationship, avg. samples". All SVD methods presented earlier in this section contained the term $u_i v_j$. The choice of multiplication as the expression connecting numeric user preferences with movie features (hidden genres) is arbitrary and should be verified (note the redundancy between changing the function connecting u_i with v_j and changing the priors for u_i and v_j). To see which function $f(u_i, v_j)$ is the proper one I will try nonparametric modelling. The method used is a two-dimensional variant of the K-means algorithm or fuzzy C-means: each user becomes assigned to one of 100 clusters, and similarly, each movie becomes assigned to one of 100 another clusters. The model used is following: each centre μ_{ab} of 100×100 rating clusters is drawn from $N(0, \sigma_\mu^2)$. Cluster $a(i)$ of each user i and cluster $b(j)$ of each movie j are drawn uniformly from the integer set 1..100. Then all ratings are drawn from $N(\mu + c_i + d_j + \mu_{ab}, \sigma^2)$. Predictions are made using i, j sampled from their posterior distributions after observing the data, and using the current iteration's estimates of μ_{ij}. The function $\mu_{ab} = g(a,b)$ modelling the relationship between clusters is estimated by a regularized average of in the group of users belonging to the cluster a, that rate movies from the cluster b:

$$\mu_{ab} = g(a,b) = \frac{(\sum_{ij \in A_{ab}} (r_{ij} - \mu - c_i - d_j)) + \mu * \lambda}{N_{ab} + \lambda}$$

with λ fixed to 100. In each subsequent iteration the clusters for users and movies are sampled using the Metropolis-Hastings method, that compares likelihood of two clusters: the previous one, and a randomly (uniformly) chosen new one. 100 iterations were run as burn-in, and additional 100 to calculate the averaged predictions. The averaged predictions gave very good RMSE 0.9498, the best among all methods listed in this section, but it should be noted, that the learned function $g(a,b)$ encapsulates more information than only one user-movie feature. Repeatedly learning the algorithm on the residuals of the same algorithm (in order to learn a new function representing more features) did not improve RMSE, likely because of overfitting.

The result of running the method for 200 iterations is a temporary assignment of users and movies to clusters and a function $g(a,b)$ with estimates of average rating in 100×100 clusters. After learning the coordinates a, b were unordered. To visualize the content of

the matrix, I chose the largest value y_{ab} in the matrix, and exchanged a to 100, and b to 100. Then I sorted the matrix rows by $g(a, 100)$ and matrix columns by $g(100, b)$, yielding a permutation of the original matrix: $A_{new} = P_1 A P_2$. In the new matrix the largest value is in the cell $g(100, 100)$, and $g(a, 100)$ and $g(100, b)$ are ordered sequences. Plot 28 shows selected columns of the permuted matrix ($g(a, b)$ was renamed as $rel(i, j)$).

A plot for the varying second coordinate b is similar to the plot 28 for index a. To save space it is skipped.

Plot 29 shows four 3D visualizations of the sorted relation $g(a, b)$, where the neighboring groups of 10 rows and 10 columns were averaged. The first chart plots averaged $g(a, b)$ in coordinates $X1, Y1$ chosen so that the averaged values of $g(a, b)$ on two edges are straight lines. The second chart uses coordinates $X2, Y2$ in which the averaged $g(a, b)$ on the other two edges are straight lines. The third and fourth (bottom) charts use coordinates U, V from SVD of $g(a, b)$ (the coordinates are averaged in groups of 10). The third chart plots the averaged $g(a, b)$ values, and the fourth chart plots the $u_a v_b$ approximation of $g(a, b)$.

We see that the learned function is close to the multiplication operation for some choice of u_a values assigned to user clusters a, and some choice of v_b values assigned to movie clusters b, but the learned function is not exactly the multiplication. After fitting parameters in several functional forms by a general purpose optimization procedure nlm() in R, and after removing low significant terms, the best approximation found of $g(a, b)$ was following:

$$\hat{r}_{ab} = \alpha + u_a v_b + \beta u_a^2 + \gamma v_b^2$$

where all parameters u_b, v_b, and a, b, c were fitted by the nlm() optimizer in R. The resulting weights were $\alpha = 0.04, \beta = -0.16, \gamma = -0.24$. The learned function $g(a, b)$ seems to be between $u_a v_b$ from the regularized SVD model, and $(u_a - v_b)^2$ from the Euclidean embedding used in [Kho10] (of course to confirm or reject this hypothesis experiments on SVD with more features are needed). The observed results justify to some degree using the multiplication in SVD-type models for the Netflix data.

Summarizing the experiments with nonparametric learning of priors and nonparametric learning of the connecting two-parameter function, the results of experiments suggest to choose different priors for u, v parameters in regularized SVD, and possibly to change the multiplication $u_i v_j$ to another function (but close to the multiplication). Whether it is possible to improve the predictive accuracy of regularized SVD type methods by following those suggestions, this is a topic for further research.

The crux subproblem appearing in regularized SVD methods is repeatedly solving a Bayesian linear regression equation with errors in predictors: $y_{ij} = u_i(\overline{v}_j + \epsilon_j) + \epsilon$ (and an analogous equation for v_j). Choices to make here are to decide on the right prior for u_i (a proper short-tailed distribution), the right form of error ϵ and errors in predictors ϵ_j (assuming here that the predictors $\overline{v}_j$ are constant), and the right computational procedure to calculate the posterior u_i. The error term ϵ may need to be modelled separately per observation ij [Tom07]. The best performing one-feature SVD method in this section was the VB approximation with the assumption of Gaussian prior for u_i, Gaussian ϵ, and Gaussian errors in predictors ϵ_j (and correspondingly, also all Gaussians for inference of v_j). More experiments on the Netflix data are needed to find out if the assumptions of the model can be improved, and if the VB approximation of the Bayesian approach gives here best possible results.

Let's summarize the whole section. I described several ways of learning the one-feature SVD model with biases $c_i + d_j + u_i v_j$, and compared the resulting RMSEs. I examined neural networks-like approaches based on minimizing a regularized cost function, and Bayesian, probability-based approaches. In the cost function minimization approach I considered various forms of regularization, and different methods of optimization: gradient descent (a first order method) and alternating jumps to minimum for subsequent parameters (a

Figure 28: Ordered columns of the learned function

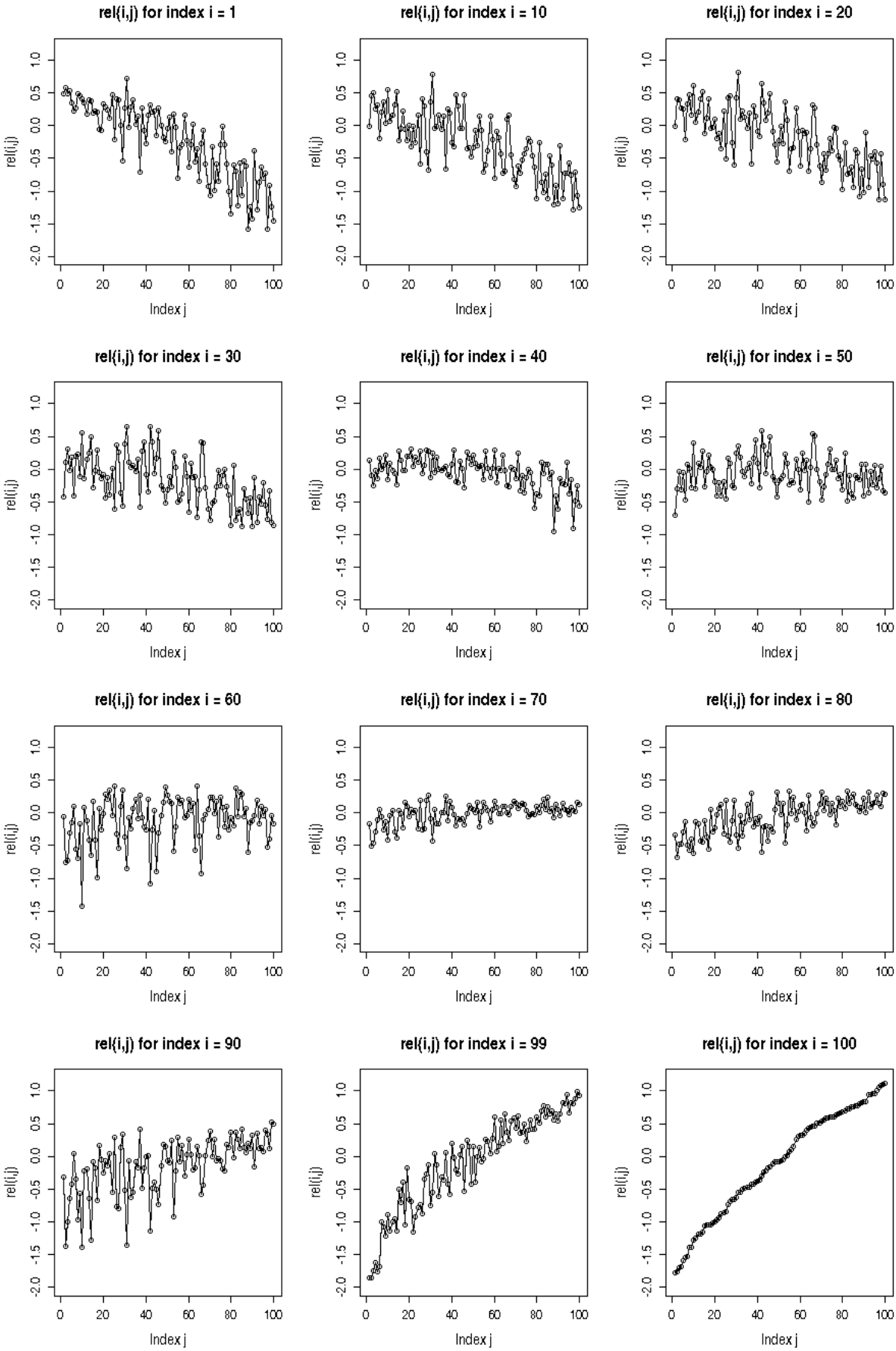
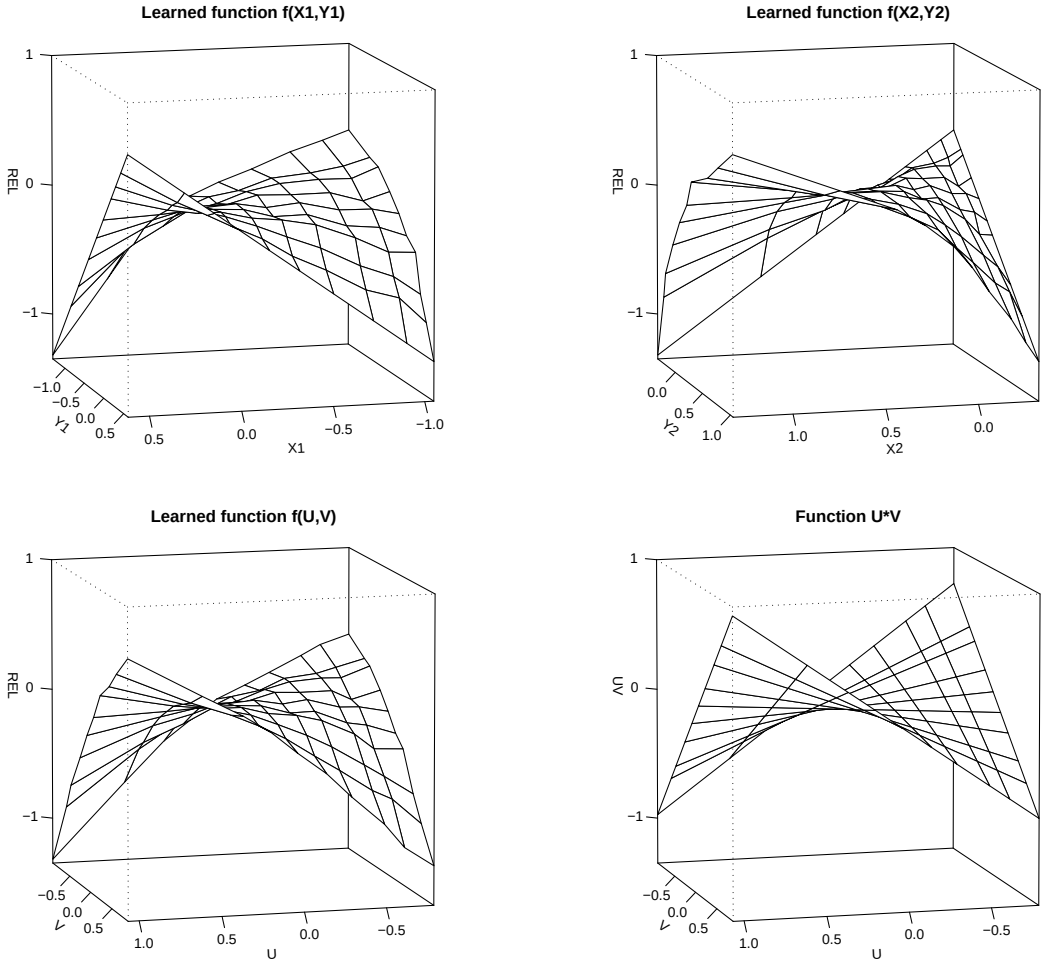


Figure 29: Visualization of the learned function



second order method). It was important to use a regularization increasing linearly with the increasing number of observations. In the Bayesian approach I used MCMC and VB approximations, assuming Gaussian priors for parameters in the model. The Gaussian assumption was verified by using a prior in nonparametric form. It turned out that the nonparametric method suggests different priors than Gaussians for u_i and v_j. I attempted also to verify, whether the multiplication $u_i v_j$ is the right operation connecting numerical user preferences with a movie feature (learned genre), and the experiment to a large degree justified the choice of multiplication.

4.3 Regularized Singular Value Decomposition

In section 4.2 I described simple models, ending with the analysis of the regularized SVD model in its simplest form with only one feature. Now I extend the analysis to the regularized SVD with $30 - 200$ features. Variants of regularized SVD, extended by using time information (see section 4.3.6), and with improved modelling of user preferences (4.3.5), were methods with the best accuracy on the Netflix Prize task.

SVD-type methods (and matrix factorization methods in general) contain a component $\mathbf{u}_i^T \mathbf{v}_j$, where the vector of real numbers $\mathbf{v}_j$ represents automatically learned features of movie j, and the vector $\mathbf{u}_i$ represents the learned user preferences for each of the corresponding features v_{jk}.

As a naming convention, I will call "regularized SVD" those of methods containing the factorization component that are similar, by the form of the model and by the result of learning the parameters, to the standard SVD from linear algebra. In particular, regularized SVD variants sort features roughly from largest to smallest magnitudes. I will call "matrix factorizations" methods that learn any rotation of $U'V'^T = UCC^TV^T$, without sorting the columns according to their magnitudes, and with the columns of U and V not necessarily close to orthogonality. I will count as matrix factorizations also methods restricting values of U and V, like non-negative MF, as well as generalized factorizations, where the output depends from the factorization component through a nonlinear link function $y_{ij} = g(\mathbf{u}_i^T \mathbf{v}_j)$. Matrix factorizations are one of the possible ways to capture subsets of matrix entries $(user, movie)$ that have the expected rating larger (or smaller, respectively) than explained (predicted so far) by global effects and by the remaining part of the model.

Before moving on to the extensive description of the full-featured regularized SVD in the next sections, let's first recall the standard SVD method known from linear algebra. Singular Value Decomposition factorizes the matrix A of size $n \times m$ into a product of three matrices: $A = U\Sigma V^T$, where U of size $n \times k$ and V of size $m \times k$ are matrices with orthonormal columns ($U^TU = V^TV = I_k$), and Σ is a diagonal matrix of size $k \times k$, $k = min(n, m)$. The values on the diagonal of Σ are sorted in decreasing order, along with the corresponding columns of U and V. SVD is related to eigenvalue decomposition: matrices A^TA i AA^T factorize into $A^TA = V\Sigma^2V^T$ and $AA^T = U\Sigma^2U^T$. Principal Components Analysis (PCA), a popular technique of statistical data analysis, which uses eigenvalue decomposition, can also be performed by SVD of the data matrix with centered columns.

If we leave only $k <= min(n, m)$ largest singular vectors in the SVD, the matrix $\hat{A}^{(k)} = U_k\Sigma_k V_k^T$ is an optimal low-rank approximation of A, as measured with Frobenius norm: the difference between $\hat{A}^{(k)}$ and A has minimal Frobenius norm $||A - \hat{A}^{(k)}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}(a_{ij} - \hat{a}_{ij}^{(k)})^2}$ among all matrices with rank at most k. This fact is convenient, because our Netflix Prize task is to minimize test RMSE, and that is Frobenius norm calculated for some subset of matrix cells. If only a small fraction of values were missing in the data matrix, standard linear algebra SVD could work well with repeated imputing the missing values with the results of the SVD approximation. The first attempts to use SVD-type algorithms for collaborative filtering were to treat missing values as zeros [Bil98], fill missing values with average item rating [Sar00], use a dense subset of data [Gol01], or to use a sparse SVD without regularization, imputing missing values with expectation-maximization-like approximations in a computationally efficient way [Bra02, Bra03, Zha05, Kur07, All10]. For the Netflix Prize data, where about 98.9% of matrix entries are missing, the problem is that imputation methods cause overfitting the data and bad prediction accuracy. Proper approximate Bayesian inference is much more accurate, leading to regularized SVD-type methods, which have roughly similar form to linear algebra SVD with sorted singular values, but lose properties like exact orthogonality of columns of U and V, and the optimization task may have local minima.

The models that are called here regularized SVD, and are described in detail in the following sections, all have the form:

$$\hat{r}_{ij} = \mu + c_i + d_j + \sum_{k=1}^{K} u_{ik} v_{jk} = \mu + c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j.$$

Similarly to the idea of centering columns in PCA, here biases are used, which are, in a sense, centering columns and rows simultaneously (more about global effects in section 4.3.1). There are many possible ways to realize learning parameters, resulting in varying predictive accuracy, running time, degree of simplicity of the algorithm, robustness, and other characteristics. The parameters u_i, v_j can be interpreted as random variables in approximate Bayes approaches (here were used Variational Bayes and MCMC methods, see section 4.3.3), or as parameters to optimize inside a regularized cost function in neural networks-like approaches (section 4.3.2). Another interesting option, not yet extensively explored, was to regularize the whole approximating matrix UV^T by a matrix norm (section 4.3.7). As for the optimization algorithms, a choice to make was either to use a first order method, like gradient descent, following the first derivative of the cost function, or a second order method, like alternating optimization of the parameters by jumping to the marginal minimum of the quadratic cost function. In all regularized SVD methods, approximate Bayesian and NN-like, there was a choice either to learn a group of all user parameters $\mathbf{u}_i$ at one time (all movie parameters $\mathbf{v}_j$, respectively), or to learn each parameter u_{ik} separately. Biases c_i, d_j were learned in a preprocessing phase, or treated as part of the model, similarly to the parameters u_i, v_j. The output of predictive models was clipped to range $1 - 5$ (it is also possible to use a smooth sigmoidal function to restrict the range of the output).

The models used usually contained a few additional parameters (regularization constants, hyperparameters, learning rates) to optimize, and a different choice of values resulted in varying accuracy measured by RMSE. In my implementations, those parameters were optimized either roughly by hand-tuning, or in a few cases automatic tuning was used, using the procedure Praxis [Bre71] from the Netlib library.

The structure of missing data has to be taken into account [Mar08], particularly the property that users tend to watch and rate movies they like. Modelling the missing data structure is probably more important for generating recommendations, but it helped also to some extent to improve prediction accuracy on the task of minimizing test RMSE (see section 4.3.5).

An additional advantage of the regularized SVD methods is the possibility of easy parallelization.

4.3.1 Global effects

When solving machine learning tasks often complex, large models are developed with thousands, or sometimes millions (as is the case with the Netflix data) parameters. Often, those large models are possible to improve by using noticed or automatically found global effects, that is patterns that are present in the entire dataset or in a large part of the dataset. An example of using global effects is a popular method of exploratory data analysis, Principal Components Analysis, in which, before calculating the principal components, columns of the data matrix are centered (the mean values of the columns are subtracted). Similarly, SVD-type methods and other algorithms applied to the Netflix data were usually improved by including global effects – most commonly used were: global mean, user biases (row effects in the data matrix) and movie biases (column effects). Comparing to column centering in PCA, using biases in the models for the Netflix data can be understood as simultaneous centering of rows and columns. Biases usually explain a large portion of data

variability. In the Netflix task, as summarized in [Dro11], 57% of variance of ratings is left unexplained, 33% is explained by best models of biases [Kor09b], and 10% explained by the model part capable of personalization, such as matrix factorizations.

The idea behind using biases and other kinds of global effects is that to model non-complex global patterns, trends or dependencies noticed in data it is sufficient to use only few parameters. More complex models, like the regularized SVD (which contains millions of parameters when applied to the Netflix data), usually are capable to model the same global effects, but modelling an effect with more parameters than necessary can cause overfitting the data. Directly modelling the noticed global effects reduces the number of model parameters needed and thus often improves predictive accuracy.

Let's list the global effects most commonly used to augment various methods for the Netflix Prize task. Proper modelling and learning of global effects was useful not only in matrix factorizations, but also in other accurate methods used in the Netflix task, like RBM or K-NN. The three most frequent, called sometimes the baseline methods, were: global mean, user and movie biases. Models using only those three effects were examined more closely in section 4.2 "Simple models". In [Bel07b] several compound global effects were proposed in the form: User $\times$ f_j, denoting learning a separate parameter for each user i, multiplied by a fixed per-movie effect f_j. The case of user bias is obtained by setting $f_j = 1$. Movie compound effects are constructed similarly: Movie $\times$ f_i, where one parameter per each movie j is learned, multiplied by a fixed per-user effect f_i. The best performing compound global effects were: User $\times$ { Movie average, Movie standard deviation, Movie support, Time(user)$^{0.5}$, Time(movie)$^{0.5}$} and Movie $\times$ { User average, User standard deviation, User support, Time(user)$^{0.5}$, Time(movie)$^{0.5}$}. Time(user) denotes time elapsed (the number of days) since the first rating of the user, and Time(movie) denotes time elapsed since the first rating given to the movie. It is an open question, how to automate searching for simple or compound global effects in data.

An additional possibility to improve accuracy was convolving the global effects with an appropriately chosen kernel expressing distance in time between two ratings of a user (see [Tos08b], part 16, "Global Time Effects" for details about this method). The paper [Tos08b] reports $\text{RMSE}_{qual} = 0.9450$, and $\text{RMSE}_{probe} = 0.9509$ for 19 global time effects learned sequentially, with each effect having 5 parameters set by automatic parameter tuning (the method of tuning was described in [Tos08b]).

[Kor09b] obtained a global effects model with $\text{RMSE}_{qual} = 0.9278$ by splitting the item bias according to the rounded logarithm of the frequency effect (frequency was defined as the number of ratings given by a user at a given day [Pio09]).

One decision to make is which effects to use. Another decision is to choose the best way of learning the parameters. In section 4.2.3 "Only biases" I compared several ways of treating the parameters of a simple model of user and movie biases, and compared several ways of optimization. For the simple model of biases the results of learning are similar for approximate Bayesian methods, such as Variational Bayes, MCMC, simple MAP point estimation, and for regularized cost function optimization (here equally well work first order methods like gradient descent, and second order methods, which boil down to alternating shrinked estimation of effects, or to alternating ridge regressions with few parameters). The experiments with nonparametric shared priors supported the assumption of approximate Gaussian priors for biases.

There are several possible ways of combining global effects with more complex models: global effects can be learned sequentially, one on residuals of another, and used as preprocessing, or GE can be learned simultaneously, and used as preprocessing of other models, or GE can be treated as a component of the larger model, and learned simultaneously with the remaining parameters of the model. As seen in section 4.2.3, simultaneous learning gives better accuracy from sequential learning, but the accuracy improvement is smaller

when using combining GE with larger models, and even smaller when comparing the ensemble accuracy. Sequential-preprocessing was used in [Fun06] to learn biases, in [Bel07b] to learn compound GE, and in [Tos08b] to learn GE with a time convolution term. GE were used as a component of the model in [Tak07a, Tak07b] (biases and other fixed effects) and in [Pat07] (biases only). In [Pot08] biases were learned simultaneously, user day bias was added, and a correction for variance of user ratings, obtaining in effect $\text{RMSE}_{qual} = 0.9488$ by using global effects only, with addition of user day bias. All three methods of learning GE parameters (sequential-preprocessing, simultaneous-preprocessing, and as part of the model) were used in the methods described later in this work.

Learning can vary depending on how we want to evaluate the method. We can either optimize predictive accuracy of global effects alone, or optimize accuracy of the larger model that is combined with the global effects (as a preprocessing or as part of the model), or learn the parameters of global effects to optimize accuracy of blended ensemble of many methods [Pio09, Tos09]. Different choices of a learning scheme can give different accuracy, computation time, or ease of tuning (manual or automatic). In this work mostly the first two evaluation criteria (accuracy of GE alone and accuracy of a larger model combined with GE) were used to choose the best GE for a method but the ensemble accuracy was a criterion of model selection, and gave hints where to focus efforts when exploring the space of possible models.

4.3.2 Neural Networks approach to SVD

In the neural-networks-like approach to SVD we minimize a regularized cost function containing the parametric form of the model, with each parameter taking a single value after minimization. The cost function minimization approach can be seen as a simplification of the Bayesian modelling, where the resulting approximate parameter inference used to calculate the posterior expected loss usually takes the form similar to some neural network. By limiting the considered methods to construing a cost function directly we are faced with fewer choices to make, in comparison to the full probabilistic modelling. It is an opportunity to save effort needed to produce a method with an accuracy good enough for a given prediction task. A drawback is that the cost function approach introduces inaccuracies. The typically considered families of cost function approaches have smaller power of expression than probabilistic models, which manifests by overfitting or underfitting, and can decrease the predictive accuracy in comparison to proper probabilistic modelling (the expressive power of neural-network-like approaches is similar to limiting ourselves to using alternating MAP estimation of parameters in probabilistic models). An important feature of the most accurate regularized cost functions for the regularized SVD model was using special regularization, nonstandard for neural networks known from the literature, for which the regularization parameter (called "weight decay") in the error cost function is constant. In the regularized SVD models used on the Netflix task the best predictive accuracy was obtained by using regularization in amount increasing linearly (or roughly linearly) with increasing number of observations.

The main decisions to be made in cost function based predictive modelling are:

- choosing the model structure – to capture (with a perspective of utility for prediction) dependencies, patterns, effects, emerging probability distributions, importance of observations (weighting observations), the form of the model output, the right representation of noise,
- choosing the error cost function, including the form of parameter regularization – either constant, like in neural networks, or linear, like it is suggested for SVD-type models by Bayesian methods, or other choices,
- how to minimize the cost function – a first order or a second order method; how to avoid local minima,

- how to tune the additional parameters, such as regularization constants for different groups of model parameters, or learning rates.

The considered cost functions, defined on a training set Tr, have the form:

$$l(\theta_1, ..., \theta_P) = \frac{1}{2} \sum_{ij \in Tr} (r_{ij} - \hat{r}_{ij}(\theta_1, ..., \theta_P))^2 + \frac{1}{2} \sum_{p=1}^{P} \lambda_p \theta_p^2 \tag{26}$$

After specifying the form of the function $\hat{r}_{ij}$, which aims to approximate ratings r_{ij} from the training set and to predict new ratings, we can learn the parameters of $\hat{r}_{ij}$ by minimizing the cost function l. One way to minimize is to use gradient descent, that is following the negated gradient of the function l:

$$\frac{\partial l}{\partial \theta_p} = - \sum_{ij \in Tr} (r_{ij} - \hat{r}_{ij}(\theta_1, ..., \theta_P)) \frac{\partial \hat{r}_{ij}}{\partial \theta_p} + \lambda_p \theta_p \qquad \theta_p \ \text{-}= \eta \frac{\partial l}{\partial \theta_p}$$

, where η is the learning rate.

The parameters or groups of parameters can be also alternately optimized by a jump to a marginal minimum by setting $\frac{\partial l}{\partial \theta_p} = 0$.

Regularized SVD methods assume the following form of rating estimation in the cost function:

$$\hat{r}_{ij} = \mu + c_i + d_j + \sum_{k=1}^{K} u_{ik} v_{jk} = \mu + c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j. \tag{27}$$

With the rating estimation by (27), the resulting cost function (26) is quadratic with respect to each parameter when the remaining parameters are fixed.

Different realizations of the neural networks SVD were tried out, with different choices of learning algorithms and regularization. The non-regularized attempts [Sar00, Kur07] overfit the data heavily and have inferior accuracy to the regularized ones. Moreover, for the Netflix dataset with 99% of data missing, sparse learning methods that ignored the missing data were more accurate (on the Netflix task) and much faster than methods using data imputation.

The most known implementation of regularized SVD in [Fun06] used linear regularization, with learning biases and each feature once. In [Fun06] first movie biases d_j are learned, then user biases c_i on residuals of d_j, both learned by a jump to the marginal minimum of cost function with a constant regularization parameter $\lambda_{cd} = 25$. For each k, the pair of features u_{ik}, v_{jk} is learned on the residuals of biases and features for all previous $k' < k$: $\hat{r}_{ij}^{(k-1)} = \mu + c_i + d_j + \sum_{k'=1}^{k-1} u_{ik'} v_{jk'}$, by minimizing the cost function with stochastic gradient descent. The regularization parameters for u_{ik}, v_{jk} are linearly dependent from the number of observations of user and movie: $\lambda_{u_{ik}} = 0.02\ N_i$, $\lambda_{v_{jk}} = 0.02\ N_j$. The stochastic gradient descent update rules for each observation (user i gives a rating r_{ij} for movie j) are following:

$$u_{ik} \mathrel{+}= \eta((r_{ij} - \hat{r}_{ij}^{(k-1)} + u_{ik} v_{jk}) v_{jk} - \lambda u_{ik})$$

$$v_{jk} \mathrel{+}= \eta((r_{ij} - \hat{r}_{ij}^{(k-1)} + u_{ik} v_{jk}) u_{jk} - \lambda v_{ik})$$

The learning rate parameter η was fixed to 0.001. The above algorithm gave $\text{RMSE}_{qual} \approx 0.91$ [Fun06]. As the author of that method pointed out later [Fun07], better results gives learning u_{ik}, v_{jk} simultaneously for all k.

A related algorithm to regularized SVD, Maximum Margin Matrix Factorization (MMMF) [Sre05], regularizes matrix factorization with the matrix trace norm. MMMF proposed in [Sre05] for collaborative filtering used hinge loss, but with MSE loss it would be equivalent to regularized SVD of a dense matrix, with a constant regularization parameter, and with the dimensionality constraint removed.

In [Pat07] biases c_i, d_j were learned simultaneously with u_{ik}, and v_{jk} for subsequent k, with special regularization of biases, obtaining $\text{RMSE}_{qual} = 0.9070$ for $K = 96$ features (without including 15% of the probe set in the training set).

The works [Tak08a, Tak08c] proposed the algorithm BRISMF (Biased Regularized Incremental Simultaneous Matrix Factorization), which is similar to [Fun06], but all parameters, including biases, are learned simultaneously. The method gave $\text{RMSE}_{qual} = 0.8962$ for $K = 100$ features.

[Sal07a] used a momentum term to speed up the gradient descent, and their implementation learned parameters in batches.

Instead of using gradient descent, another method of optimizing the cost function (26) was marginal optimization of single parameters or groups of parameters by jumping directly to the minimum of the cost function, with the remaining parameters fixed. The following methods optimized groups of features at a time – all preferences of a given user, and all features of a given movie. [Bel07d] optimized (26) with non-negative least squares. [Pat07] used one-time ridge regression to post-process gradient descent SVD (nonlinear kernel ridge regression worked better). [Zho08] optimized the cost function (26) with alternating ridge regression. [Pio09] used ridge regression with a special diagonal regularization matrix:

$$u_i = (V_i^T V_i + (\lambda_1 + \lambda_2 N_i)\overline{V})^{-1}(V_i^T Y + \alpha\overline{u})$$

where V_i are features for movies rated by user i, $\overline{u}$ is a weighted average u_i from the previous iteration, weighted by the user support. $\overline{V}$ is a diagonal matrix containing diagonal values of $V_i^T V_i$, averaged for all users i.

Generally, gradient descent learning resulted in converging to better minima than alternating ridge regression of groups of parameters, but accuracy and convergence of alternating ridge regression improve largely when using a good prior estimate of user preferences – see section 4.3.5. [Cic09] observed similar difficulty in alternating minimization in non-negative matrix factorizations, and [Mis11] proposes algorithms for learning a factorized distance matrix, that circumvent similar difficulties of second order optimization methods.

The cost function (26) can also be optimized one parameter at a time by a jump to the local minimum, and this learning method in turn gives similar accuracy to gradient descent learning. A method of this type was proposed in section 4.3 of [Bel07a]. Instead of using shrinking estimation with one variable ridge regression, special weighting of residuals was used in [Bel07a] in the form of shrinking the residuals, a method close to one variable ridge regression with constant regularization. Optimizing one parameter at a time was also used in [Pil10].

Throughout the work I implemented several variants of regularized SVD, starting from multiple variants of gradient descent SVD with biases [Pat07], through alternating ridge regressions with various forms of regularization, learning one parameter at a time or multiple parameters, to approximate Bayesian learning. Most experiments with SVD after the previous publication [Pat07] were attempts to approximate the Bayesian model [Tom07] (described in the next section 4.3.3) and its Variational Bayesian version. The most accurate SVD methods in my experiments, among the cost function based approaches, used constant-linear regularization and learning one parameter at a time, with each feature fully learned (15-20 iterations) before learning the rest, and with all iterations repeated 2-4 times. The methods SVDM and SVDMT, described in section 4.3.6 "Time effects", were time-enhanced variants of regularized SVD in the cost function approach, with improved user preferences, and with the regularization enhanced by a shared prior variance of user parameters. The earlier mentioned SVD variants [Fun06, Pat07, Tak07a, Bel07d, Zho08] used linear regularization $\lambda_u N_i$ for users and $\lambda_v N_j$ dla movies, which is better than regularization with constants λ_u, λ_v, but inferior to linear regularization with an additional constant term $\lambda_{u1} + \lambda_{u2} N_i$ for user i, who rated N_i movies, and $\lambda_{v1} + \lambda_{v2} N_j$ for a movie j

rated N_j times (the constant-linear form of regularization is suggested by the approximate Bayesian inference, see section 4.3.3). The regularization parameters in my implementations were most often tuned automatically by the Praxis procedure from the Netlib library, or sometimes tuned by hand.

The method SBRISMF [Tak08b, Pil09d] adds to the regularization additional terms beyond the constant and linear: $\sqrt{N_i}$ and $N_i/\log N_i$. With automatic parameter tuning of all regularization parameters (seven times more regularization parameters than BRISMF), $\text{RMSE}_{quiz} = 0.8905$ was obtained for $K = 1000$ features.

An advantage of using the same dataset and evaluation criterion is the possibility to compare solutions of different authors. Table 14 lists various neural-networks-like SVD implementations of various authors, with accuracy measured by RMSE_{quiz} on the Netflix Prize dataset.

Table 14: Regularized SVD/MF. Experimental results.

Method	Regularization	Learning	Grouping	K	RMSE_{15}	RMSE_{quiz}
Incremental SVD [Fun06]	$\lambda_{uv} = 0.04n$, $\lambda_{cd} = 25$	gradient	single	40		0.91
BRISMF [Tak08a, Tak09a]	$\lambda_{uvcd} = 0.04n$	gradient	multi	100		0.8962
IncFctr [Bel07c]	$\lambda_{uv} = 0.05n$, $\lambda_{cd} = 25$	alt. min.	multi	40		0.9135
SimuFctr [Bel07a, Bel07c]	$\lambda_{uv} = 0.05n$, $\lambda_{cd} = 25$	alt. min.	multi	20		0.9056*
NNMF [Bel07b, Bel07c]	$\lambda_{uv} = 0.05n$, $\lambda_{cd} = 25$	alt. min.	multi	40		0.9094
ALS [Zho08]	$\lambda_{uv} = 0.05n$	alt. min.	multi	300		0.9017
BasicSVD [Tos08b]	$\lambda_{uv} = 0.05n$	alt. min.	multi	300		0.9028
SBRISMF [Tak08b]	$\lambda_{uvcd} = 0.02n$	gradient	multi	1000		0.8905
BRISMF250 [Pio09]	$\lambda_{uvcd} = 0.02n$	gradient	multi	100		0.8945
SSVD-04-10 [Pio09]	$\lambda_{uvcd} = 0.02n$	alt.min.	multi	10		0.9236
RSVD (Incremental SVD)	$\lambda_{uv} = 0.02n$, $\lambda_{cd} = 0.02n$	gradient	single	96	0.9155	($\sim$ 0.9185)
RSVD+BASIC [Pat07]	$\lambda_{uv} = 0.02n$, $\lambda_{cd} = 0.02n$	gradient	single	96	0.9094	($\sim$ 0.9125)
RSVD2	$\lambda_{uv} = 0.02n$, $\lambda_{cd} = 0.02n$	gradient	single	96	0.9064	($\sim$ 0.9095)
RSVD2+BASIC [Pat07]	$\lambda_{uv} = 0.02n$, $\lambda_{cd} = 0.02n$	gradient	single	96	0.9039	0.9070**
SVD1N1	$\lambda_{uv} = 2$	gradient	single	30	0.9110	($\sim$ 0.9140)
SVD1N9A	$\lambda_{uv} = 2$	gradient	single	85	0.9072	($\sim$ 0.9100)
SVD1N19	$\lambda_{uv} = 2$	gradient	single	85	0.9073	($\sim$ 0.9100)
SVDRR [Pat07]	$\lambda_{uv} = 8$	min.	multi	96	0.9088	($\sim$ 0.9110)
SRR4	$\lambda_{uv} = 8$	min.	multi	96	0.9119	($\sim$ 0.9150)
SRR5	$\lambda_{uv} = 8$	min.	multi	96	0.9128	($\sim$ 0.9160)

* - training set without probe set

** - training set without 15% of probe set

RSVD, RSVD2, BASIC, and SVDRR were described in [Pat07]. BASIC is a set of six predictors: empirical user probabilities, and movie mean. RSVD is the original regularized SVD [Fun06] with global effects by BASIC. RSVD2 is regularized SVD with biases. SVD1N1 is RSVD2 with less features and changed number of iterations. SVD1N9A is RSVD2 with regularization (shrinking) of the sum of movie features. SVD1N19 is RSVD with nonlinear postprocessing. SVDRR is RSVD2 postprocessed by ridge regression on normalized movie feature vectors. SRR4 is RSVD2 postprocessed by ridge regression, and SRR5 is RSVD2 postprocessed by approximate ridge regression. In the listed variants of RSVD and RSVD2, features were learned one time, but with biases in RSVD2 learned simultaneously with features. This way of learning was inferior to repeating 2-3 times learning of all features, or to simultaneous learning of all features (both used in matrix factorization variants in my later experiments). All of the above methods were parts of the ensemble in [Pat07] (that paper described the most contributing methods of the 56 predictors in the ensemble), and made also minor contributions to the newer ensemble, listed in chapter 5 "Experimental results".

We have seen, that the cost function (26), (27) of the regularized SVD can be optimized

in many different ways, resulting in different accuracy. As experiments in section 4.2.5 showed, the priors of the regularized SVD parameters, usually for simplicity assumed to be Gaussians, should really be different distributions. The used cost function is an inaccurate approximation of Bayesian inference in an inaccurate probabilistic model, and optimizing it in different ways with early stopping results in ending up in different areas the hold-out test error, with the data underfitted or overfitted. RMSE accuracy is affected by the number of iterations, the order of learning parameters or groups of parameters, choice of learning rates, size of batches, initial values of parameters.

Rotating the matrix of item features gives equivalent matrix factorizations $UV^T = U'RR^TV'^T$, and different ways of optimization results in different factorizations. Learning the features sequentially, one k-th feature at a time (even with re-learning features several times on the residuals of the remaining ones), results in a factorization similar to linear algebra SVD, where the singular values are sorted from the largest magnitudes (see section 4.3.4 for an attempt to explain the meaning of the most significant features). In turn, learning all features simultaneously can learn any rotation. Because the cost function can have multiple local minima (also, marginal minimization of groups of features can cause the optimization algorithm to get stuck), and because all algorithms use early stopping, when the algorithm decides in subsequent iterations on different rotations, it influences the final values of parameters and RMSE accuracy.

On the basis of own experiments I summarize in table 15 advantages and disadvantages of different choices from the perspectives of predictive accuracy, amount of computation, and with my judgement of usage convenience in a recommender system.

Table 15: Summary of different approaches to matrix factorization.

Learning	Grouping	Predictive accuracy	Computing time	usage convenience
gradient	single	good	slow	average
gradient	multi	good	fast	average
alt.min. and VB	single	good	average	good
alt.min. and VB	multi	average	average	average
MCMC	single	good	slow	average
MCMC	multi	good	slow	average

The "alt.min., single" method denotes the mentioned earlier method with constant-linear regularization and learning one feature at a time, with learning of all features repeated three times. The MCMC and Variational Bayes (VB) methods are described later in section 4.3.3.

A disadvantage of gradient descent methods is the need to additionally tune learning rates. Too small learning rates increase computation time, and too large learning rates cause the algorithm to diverge. Another disadvantage of gradient descent methods is that if there are regularities in the input data, the parameters can be periodically biased during learning, for example, in my implementations users and movies were sorted by frequency, and the set of movie features differed at the time of learning parameters of different groups of users.

As for the computational complexity, the number of iterations in gradient descent SVD was usually $100 - 250$ (it is possible to reduce it to only a few iterations, using variable learning rates [Tak08a, Kul09]). In the gradient descent methods that learn all features at a time the computational complexity is $O(LNK)$, where L is the average number of iterations, N is the amount of data – the number of ratings observed (100 million), and K is the number of features to learn (typically $30 - 200$). In the gradient descent methods that learn one feature at a time the limiting factor is reading all values in the matrix

LK times, and the number of iterations increases, because learning of features has to be repeated several times on residuals of the remaining ones. In some cases a limiting factor in stochastic gradient descent methods can be reading and writing LN times to the matrices of parameters U and V.

Second order methods that learn all features at once have the complexity $O(L * (M * K^3 + P * K^3 + N * K))$, where M is the number of users $(480,189)$, P is the number of movies $(17,770)$, and the number of iterations L is typically about 15. For users who rated few movies, instead of ridge regression, which requires storing and inverting a matrix $K \times K$ one can use kernel ridge regression inverting a matrix $M_i \times M_i$, where M_i is the number of ratings of user i [Pil09a]. When learning one feature at a time with a second order method (alternating jumps to the marginal minimum of the cost function), and fully learning one feature before moving on to the next one, the complexity is $O(L_2 * L * (M * K + P * K + N * K))$, where the number of iterations L is $10 - 20$, and the entire learning process is repeated L_2 times, usually $2 - 4$ times, with every feature learned on the residuals of the rest of the model.

The calculations can be easily sped up by parallelization, because the tasks of calculating user preferences are independent of each other (except that they read the same movie features), similarly as the tasks of movie features are independent (except reading the same user preference vectors). One easy way of parallelizing is using the "parallel for" instruction in the OpenMP framework.

Summarizing the topic of neural-networks-like regularized SVD, the most important observation is that the constant-linear regularization $\lambda_1 + \lambda_2 n$ is better than the linear term alone λn, and the linear regularization in turn is much better than the constant regularization λ. This fact is explained by the form of approximate Bayesian inference, presented in the next section. The constant-linear regularization term may be useful in other neural-network-like applications, where it may work better than the typical in neural networks choice of constant regularization.

Because the approximated underlying probabilistic model is inaccurate for the gathered rating data (see section 4.2.5), identifying the underlying model better can possibly lead to neural-networks-like simplified methods with even better accuracy.

4.3.3 Approximate Bayesian approach to SVD

The previous section described cost function optimization methods, which are rough approximations (even with special regularization) of Bayesian inference, and without precisely specifying the probabilistic model. In this section I describe more direct and more accurate approximations of a fully specified model: MCMC with Gibbs sampling, and Variational Bayes approximation.

The idea of using MCMC (Monte Carlo Markov Chain) for learning the parameters of matrix factorization for the Netflix Prize task was first proposed in [Har07], where the author described a method that uses Gibbs sampling and empirical Bayes estimation of priors in a hierarchical model, without details of the model like the form of hyperpriors, usage of biases, etc. The author reported that, after running the algorithm for three weeks, the averaged samples yielded RMSE_{quiz} 6.3% better than Cinematch.

Variational Bayesian approaches to SVD were proposed in [Lim07] and [Rai07] for sparse matrices, and earlier in [Bis99b] for dense matrices. VB and MCMC for dense matrices were also mentioned in [Bis99a].

My experiments with approximate Bayesian approaches were based mainly on knowledge of the John Tomfohr's model [Tom07], which learned the parameters of regularized SVD model using MCMC with Gibbs sampling. The Tomfohr's model, called "Bayesian PCA", was not published, but its efficiency was demonstrated by submitting $\text{RMSE}_{quiz} = 0.8805$ of a single method to the Netflix Prize evaluation in 2007 (compare with the lead-

ing team's ensemble in the year 2007 with the best method having $\text{RMSE}_{quiz} = 0.8888$ [Bel07c]). JT's Bayesian PCA alternated between sampling the model variables, one variable at a time, according to their conditional posterior distributions $p(u_{ik}|D, \boldsymbol{\theta} \backslash u_{ik})$, $p(v_{jk}|D, \boldsymbol{\theta} \backslash v_{jk})$. To save space I skip the inference equations for u_{ik} and v_{jk}, which are analogous to the MCMC model described earlier in section 4.2.5 "One-feature regularized SVD with biases", but with the residuals including the remaining sampled features $k' \neq k$ subtracted from the rating:

$$y_{ij}^{(k)} = r_{ij} - c_i - d_j - \sum_{k'=1, k' \neq k}^{K} u_{ik'} v_{jk'}$$

JT's Bayesian PCA also contained also time effects, such as user day bias, and modelled the output variance parameter σ_{ij}^2 (the variance of the error term in the model) differing between users and movies.

The models [Har07, Sal08, Mac10, Sha10] realized approximate matrix factorization by alternately sampling whole vectors of features in the MCMC algorithm. The basic SVD model in both MCMC and VB methods was:

$$r_{ij} \sim N(\mu + c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j, \sigma^2)$$

$$\mathbf{u}_i \sim N(\overline{\mathbf{u}}, S_u) \qquad \mathbf{v}_j \sim N(\overline{\mathbf{v}}, S_v)$$

Some models assumed $\overline{\mathbf{u}} = \overline{\mathbf{v}} = 0$. The matrices S_u and S_v from the shared priors were either assumed to be diagonal, or dense, and either flat hyperpriors or inverse Wishart hyperpriors were used. A necessary assumption is constraining one of the matrices S_u and S_v, for example, fixing it to the identity matrix, or repeatedly normalizing the matrix during running the algorithm. If both matrices are unconstrained, usually the values in one matrix tend to grow to infinity, and in the second one decrease to zero. Models proposed by different authors also vary by the choice of the method of learning biases.

The MCMC algorithm with point estimation of hyperparameters (the method can be called Monte Carlo empirical Bayes) iteratively repeats the following:

- sample from the posterior distribution $p(U|R, V, \boldsymbol{\theta}_U)$,
- point estimation of hyperparameters $\boldsymbol{\theta}_U$ of the prior $p(U|\boldsymbol{\theta}_U)$
- sample from the posterior distribution $p(V|R, U, \boldsymbol{\theta}_V)$
- point estimation of the hyperparameters $\boldsymbol{\theta}_V$ of the prior $p(V|\boldsymbol{\theta}_V)$

where $\boldsymbol{\theta}_U = (\overline{\mathbf{u}}, S_u)$, and $\boldsymbol{\theta}_V = (\overline{\mathbf{v}}, S_v)$.

The conditional posterior distribution $p(U|R, V, \boldsymbol{\theta}_U)$, having sampled V and point estimated prior for U, is calculated using Bayes rule:

$$p(\mathbf{u}_i|R, V, \boldsymbol{\theta}_U) \propto p(\mathbf{r}_i|\mathbf{u}_i, V) p(\mathbf{u}_i|\theta_U) = N(E\mathbf{u}_i, Cov\ \mathbf{u}_i)$$

$$E\mathbf{u}_i = (V_{(i)}^T \mathbf{y}_i + S_u^{-1} \overline{\mathbf{u}})\ Cov\ \mathbf{u}_i$$

$$Cov\ \mathbf{u}_i = \left(\frac{1}{\sigma^2} V_{(i)}^T V_{(i)} + S_u^{-1} \right)^{-1}$$

where $V_{(i)}$ is the matrix of sampled features for movies rated by user i. Calculating the conditional posterior of V is analogous, with $V_{(i)}$ switched to $U_{(j)}$ (the matrix of sampled user preferences for users who rated movie j).

Variational Bayesian learning, in turn, assumes independence of groups of parameters (in our case U and V) in the posterior distribution, and approximates the posterior with that constraint by minimizing the Kullback-Leibler divergence of the approximation from the true joint posterior [Bis06]. In the resulting algorithm [Lim07, Bis99b], instead of sampling U and V as MCMC, the expected value is calculated: $E_V \log p(U|R, V)$, alternately

with $E_U \log p(V|R,U)$. The resulting formulas for posterior distributions are very similar to the MCMC method, only instead of sampled U and V, their expected values are used, and an additional component appears summing variances $Cov\ \mathbf{v}_j$ and $Cov\ \mathbf{u}_i$:

$$E\mathbf{u}_i = (EV_{(i)}^T \mathbf{y}_i + S_u^{-1}\overline{\mathbf{u}})\ Cov\ \mathbf{u}_i$$

$$Cov\ \mathbf{u}_i = \Big(\frac{1}{\sigma^2} EV_{(i)}^T EV_{(i)} + \frac{1}{\sigma^2}\sum_{j\in J_i} Cov\ \mathbf{v}_j + S_u^{-1}\Big)^{-1}$$

$$E\mathbf{v}_j = (EU_{(j)}^T \mathbf{y}_j + S_v^{-1}\overline{\mathbf{v}})\ Cov\ \mathbf{v}_j$$

$$Cov\ \mathbf{v}_j = \Big(\frac{1}{\sigma^2} EU_{(j)}^T EU_{(j)} + \frac{1}{\sigma^2}\sum_{i\in I_j} Cov\ \mathbf{u}_i + S_v^{-1}\Big)^{-1}$$

The components containing $Cov\ \mathbf{v}_j$, $Cov\ \mathbf{u}_i$ grow linearly with the number of observations for a user, or a movie, which justifies the use of a linear regularization component in the most accurate neural-networks-like SVD variants. (see the last section 4.3.2, and section 4.2.5).

The assumption of independence of posterior U and V in VB is reasonable, but the choice of minimizing KL-divergence is disputable. As discussed in section 4.2.5, an inconsistency in the used model and the VB with KL divergence is in that for the output variable r_{ij} assumed as Gaussian, we assume priors for u_{ik}, v_{jk} (the variables multiplied in the model) being Gaussians, and the approximated posterior distributions of u_{ik}, v_{jk} are also Gaussians. This way the output variable r_{ij} is approximated with a sum of "spike" variables – normal product distributions.

On fully observed matrices VB matrix factorization has an analytic solution, if assuming independence of features (columns of U and columns of V) in the posterior distribution [Nak11a, Nak11b].

Alternating MAP point estimation was also tried for probabilistic models similar to the above model [Tip99, Row98, Bis99a, Bel07c, Bel07d, Sal07a], with inferior accuracy to both approximate Bayesian SVD and neural-networks-like SVD with proper regularization. Expectation-Maximization learning, where user features are treated as in VB, and movie features as in MAP method, also gave worse accuracy than the best obtainable [Rob10, McM96, Can02].

Table 16 lists results of Bayesian SVD obtained by various authors. I implemented several variants of VB SVD, learning one or multiple features at a time, with different choices of priors, but they did not improve the ensemble in presence of the JT's Bayesian PCA. Some implemented variants of neural-networks-like SVD were heavily inspired by the Bayesian learning, as the SVDMT2 method, described in section 4.3.6 (see also chapter 5).

Table 16: Approximate Bayesian SVD/MF. Experimental results.

Method	Grouping	Learning	K	RMSE_{15}	RMSE_{quiz}
MCMC [Har07]	multi	MCMC, EB	256		0.8937
VB [Lim07]	multi	VB	30		($\text{RMSE}_{probe} = 0.9141$)
Simu/GaussFctr [Bel07c, Bel07d]	multi	MAP	60		0.8998
Bayesian PCA [Tom07]	single	MCMC	60		0.8805*
VB-PCA [Rai07]	single	VB, gradient	15		($\text{RMSE}_{probe} = 0.9180$)
PMF [Sal08, Sal07b]	multi	MAP	40		0.9170
BPMF [Sal08]	multi	MCMC	100		0.8965
BSRM [Zhu08]	multi	MCMC	100		0.8930
VBPCAd [Ili08]	single	VB, gradient	50		0.8990

* - the model uses time data, and models the output variance σ_{ij}^2 of observations (i,j)

** - training without probe set

I described the methods realizing the basic probabilistic matrix factorization model. They can be further improved by using time effects (section 4.3.6), better priors on user preferences (section 4.3.5), and using more refined global effects than the simple biases (section 4.3.1).

4.3.4 Examination of learned parameters

Matrix factorization methods perform dimensionality reduction and represent movies in the form of a short vector of learned parameters, which can be understood as automatically learned hidden genre representation. One might wonder, what is the relationship between the automatically learned genres and the genres distinguished by humans, such as action movie, comedy, horror, etc. The learned features can represent unnamed genres or may be a rotation, or other linear combination, of features intuitively understood by humans. For example, if movies were represented in two dimensions, say, on a plane spanned by two genres: "action" on X axis and "comedy" on Y axis (without defining those genres precisely), the basis vectors can be rotated and the movies can be represented in coordinates (1*action - 1*comedy, 1*action + 1*comedy), forming an equivalent matrix factorization after analogously rotating the user features. Different learning algorithms for sparse, regularized matrix factorization can result in different rotations of the solution, leading also to small differences in predictive accuracy. If features are learned sequentially, with each feature fully learned before moving on to learning the next ones, in the resulting factorization features will be ordered from the largest magnitudes, similarly as in the standard SVD algorithm from linear algebra, where features (singular vectors) are sorted according to their singular values.

I attempted to interpret a few features with the largest magnitudes as new movie genres. Features come from a VB version of regularized SVD (see section 4.3.3) with 32 features, fully learning one feature at a time, and with learning all features repeated three times. The experiments were backed by examining the standard genre classification and lists of keywords, both coming from the IMDb database correlated with the Netflix titles. It turned out, that the most meaningful SVD features encode concepts such as personality types, value systems, emotions, etc.

Figures 30 and 31 show the learned 32-element vectors of parameters $\tilde{\mathbf{v}}_\mathbf{j}$ for 35 example movies and TV series, chosen among the 4000 most frequently rated in the Netflix dataset. The vectors $\tilde{\mathbf{v}}_\mathbf{j}$ are outputs of a VB SVD, normalized to length one (normalized features were used in all experiments in this section). Next to every movie on the left side, two movies are shown on the right side for comparison: one positively correlated, and one negatively correlated (in a sense of the dot product of the normalized vectors $\tilde{\mathbf{v}}_\mathbf{j}^T \tilde{\mathbf{v}}_{\mathbf{j_2}}$).

Table 17 shows maximal and minimal values of the first six features among the 35 movies listed earlier in figures 30 and 31. Because of the chosen sequential way of learning the SVD model, the first features explain most of the variability in data. They correspond more or less to the largest singular values in a linear algebra SVD (but not precisely, because there is no orthogonality property, due to the presence of missing data and the use of regularization). The column "Rank" in the table 17 denotes the frequency rank – the ordering when the movies are sorted by support (by the number of ratings).

Figure 30: Example movies with their learned feature values, part 1

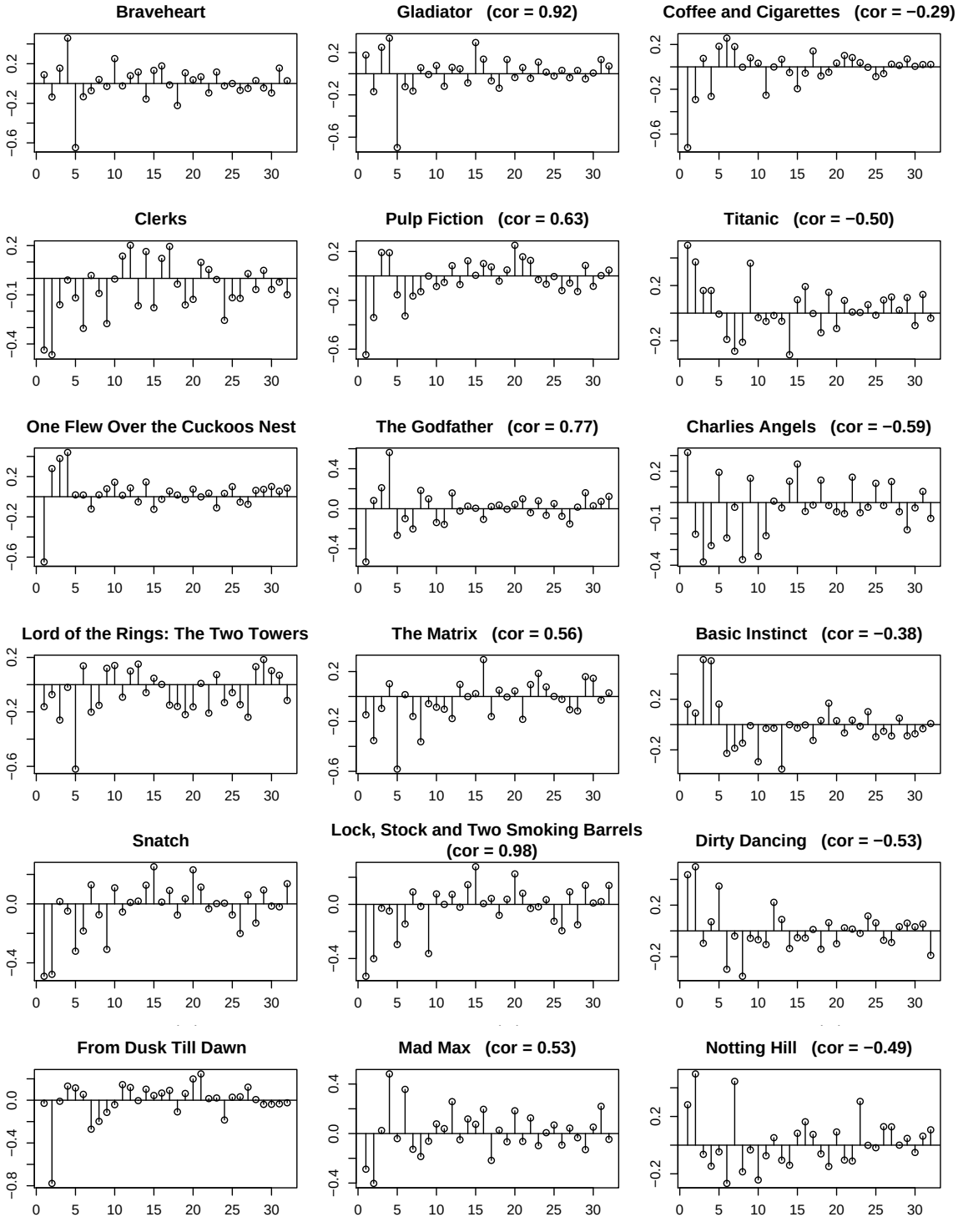


Figure 31: Example movies with their learned feature values, part 2

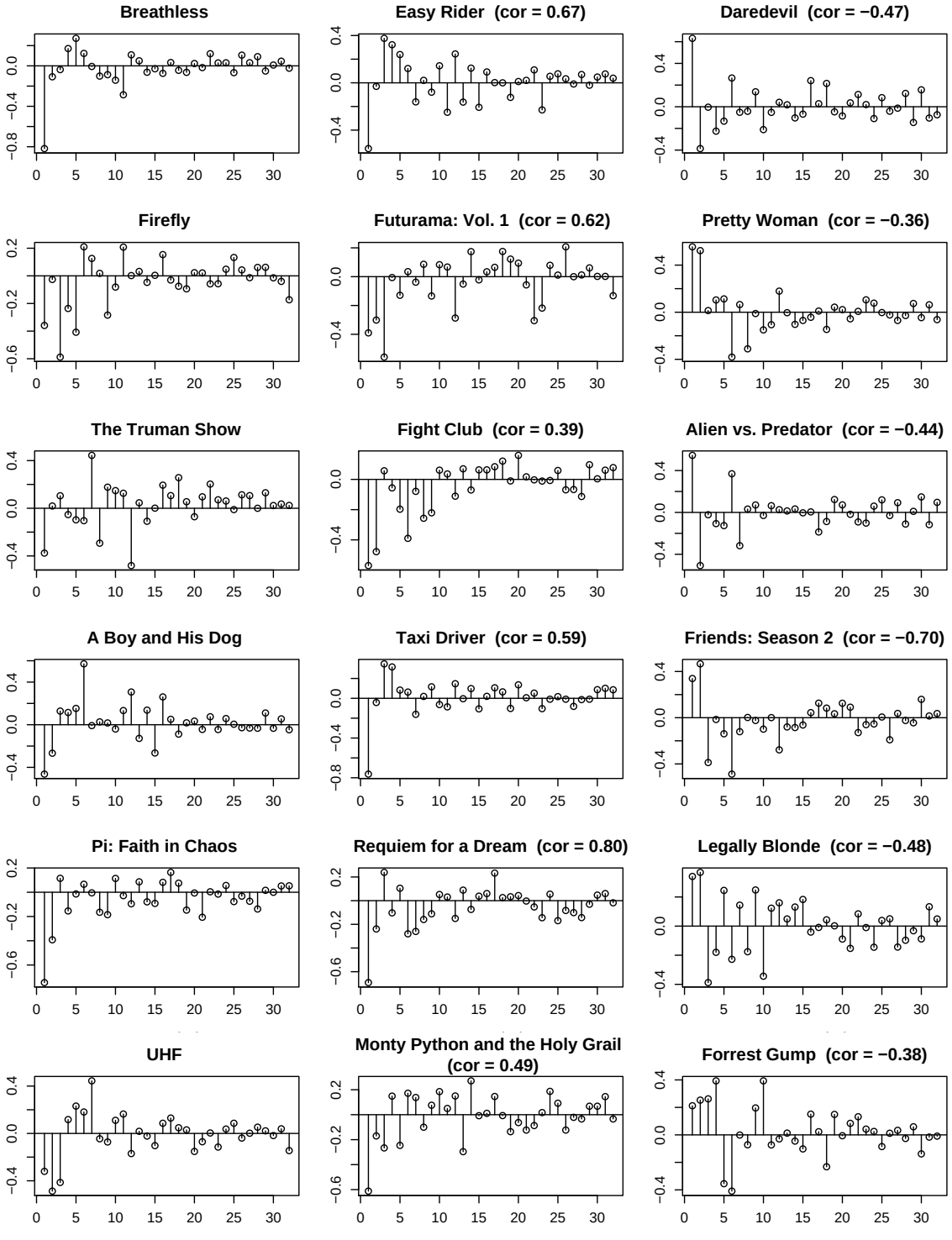


Table 17: Minimal and maximal values of features, among 4000 most popular movies.

Feature 1	Rank	Movie
0.6310	373	Daredevil
0.5550	6	Pretty Woman
0.5450	568	Alien vs. Predator
0.4920	44	Titanic
...	...	...
-0.7210	1731	Coffee and Cigarettes
-0.7450	913	Pi: Faith in Chaos
-0.7630	255	Taxi Driver
-0.8180	2587	Breathless

Feature 4	Rank	Movie
0.5630	130	The Godfather
0.5050	381	Basic Instinct
0.4820	739	Mad Max
0.4590	60	Braveheart
...	...	...
-0.2250	373	Daredevil
-0.2360	1947	Firefly
-0.2640	1731	Coffee and Cigarettes
-0.2750	491	Charlies Angels

Feature 2	Rank	Movie
0.5230	6	Pretty Woman
0.4980	45	Dirty Dancing
0.4970	337	Notting Hill
0.4660	878	Friends: Season 2
...	...	...
-0.4780	333	Snatch
-0.4870	2241	UHF
-0.5080	568	Alien vs. Predator
-0.7780	946	From Dusk Till Dawn

Feature 5	Rank	Movie
0.3480	45	Dirty Dancing
0.2720	2587	Breathless
0.2450	114	Legally Blonde
0.2390	935	Easy Rider
...	...	...
-0.5820	48	The Matrix
-0.6200	30	LotR: The Two Towers
-0.6480	60	Braveheart
-0.6990	32	Gladiator

Feature 3	Rank	Movie
0.5140	381	Basic Instinct
0.3800	227	One Flew Over the (...)
0.3760	935	Easy Rider
0.3470	255	Taxi Driver
...	...	...
-0.3880	878	Friends: Season 2
-0.4140	2241	UHF
-0.5580	1714	Futurama: Vol. 1
-0.5880	1947	Firefly

Feature 6	Rank	Movie
0.5710	3896	A Boy and His Dog
0.3690	568	Alien vs. Predator
0.3570	739	Mad Max
0.2650	373	Daredevil
...	...	...
-0.3810	6	Pretty Woman
-0.3900	85	Fight Club
-0.4090	7	Forrest Gump
-0.4880	878	Friends: Season 2

The next table 18 shows my interpretation of the first six automatically learned features in the regularized SVD, that explain most of the variability of ratings.

Table 18: Interpretation of the first six features.

Feature	Large positive value	Large negative value
1	good defeats evil, idealized world	talking, thinking, emotions, realism
2	love, safety	action, pace, surprise
3	individualism, untrustworthy environment	fairy tale
4	violence, aggression, men's world, patriarchy	independent women, feminism, matriarchy
5	gentle manner, innocence, anxiety	fight, heroism, courage
6	science-fiction, quest, journey	youth, growing up, friendship

We could also try to name the positive and negative part of a feature with single words. My attempt to name the first six features: Idealization vs. Realism, Safety vs. Surprise, Distrust vs. Fairy Tale, Testosterone vs. Feminism, Innocence vs. Heroism, Journey vs. Growing Up.

The above interpretation of features was created based on observation of sorted lists of movie titles. I performed an additional experiment that confirmed to some extent the proposed interpretation – table 19 lists the keywords from IMDb database that are the best predictors of the first six SVD features, according to a chosen criterion. Each feature was predicted by ridge regression with a regularization constant $\lambda = 10$, with predictors

chosen by greedy feature selection among 21 genres and 579 keywords from the IMDb that appear at least 50 times among 2000 movies with the largest support in the Netflix database. The scoring criterion for features in the greedy feature selection was the sum of variance explained and a fraction of the correlation between the IMDb feature and the predicted SVD feature: $\sum_i(\hat{y}_i - \overline{y})^2 / \sum_i(y_i - \overline{y})^2 + 0.05\ |Cor\ (X_k, Y)|$. Each of the six features from the regularized SVD was predicted by 30 IMDb features (genres and keywords) chosen by the greedy feature selection.

Table 19: First six SVD features labeled by IMDb keywords.

Feat.	Var. Expl.	Positive coefficient	Negative coefficient
F1	46%	Action, Family, Thriller, Romance, helicopter, chick-flick, box-office-flop, beautiful-woman	independent-film, cult-favorite, Drama, black-comedy, satire, singing, famous-score, surrealism, Documentary, extramarital-affair, cigarette-smoking, nudity, writer, critically-acclaimed, cafe, adultery, voice-over-narration, drug-use, 1950s, sex, hotel, female-nudity
F2	53%	Drama, Romance, blockbuster, friendship, female-protagonist, Family, male-female-relationship, based-on-novel, 1930s, musician, family-relationships	cult-favorite, Thriller, Action, blood, crude-humor, Horror, Comedy, shot-to-death, topless-female-nudity, sequel, shot-in-the-chest, Sci-Fi, violence, falling-from-height, blood-spatter, gore, masturbation, person-on-fire, punched-in-the-face
F3	53%	Drama, Thriller, murder, sex, neo-noir, adultery, based-on-novel, death, psycho-thriller, gun, Crime, female-nudity, box-office-flop, husband-wife-relationship, beating, shot-in-the-head	Comedy, Family, cult-favorite, Adventure, Animation, martial-arts, Fantasy, Musical, spoof, sequel, lifting-someone-into-the-air, magic, anti-hero, monkey
F4	38%	blockbuster, cult-favorite, famous-score, shootout, famous-line, Sport, revenge, Western, first-of-series, automobile, Crime, racial-slur	flashback, mother-daughter-relationship, Romance, cell-phone, box-office-flop, dancing, Fantasy, tears, female-protagonist, writer, Comedy, love, father-daughter-relationship, drink, homosexual, drinking, painting, graveyard
F5	36%	Comedy, female-protagonist, Horror, teen-angst, Music, Family	Action, no-opening-credits, honor, blockbuster, shot-in-the-forehead, slow-motion, warrior, History, epic, surprise-ending, battle, Adventure, sword-fight, explosion, redemption, cia, monkey, shot-in-the-chest, space-craft, world-war-two, hero, sword, shot-to-death, bravery
F6	38%	Action, Sci-Fi, Adventure, horse, box-office-flop, Thriller, Fantasy, world-war-two, spacecraft, pursuit, alien, murder, Family, future, outer-space	Comedy, boyfriend-girlfriend-relationship, coming-of-age, crude-humor, marijuana, gay-slur, sex, high-school, cocaine, roommate, teen-movie, friendship, obscene-finger-gesture, racial-slur, teen-angst

As we see in table 19, a relatively large amount of variance was explained by binary IMDb tags, although perhaps more tags are needed to describe features 4, 5, 6. The listed values of variance explained were estimated on the training data, and are heightened due to overfitting.

When "folksonomy" tagging is used (tags edited by community, as in IMDb), the quantity and quality of tags increase with movie popularity. For the 2000 most popular movies from the Netflix dataset, as we see from the amount of variance explained, the tags predict the SVD features well. Tags can be used, for example, to produce good priors for item features in SVD-type algorithms, which can be useful in cold start situations for items, often encountered in recommender systems (see the use of IMDb tags to predict features of movies without any ratings in the Netflix database – described in section 4.7). But for movies with more ratings, such as the movies in the Netflix Prize dataset, experiments gave counter-intuitive results, that augmenting the movies with metadata has no effect on accuracy – a small number of ratings is more informative than any amount of metadata [Pil09b, Lee08].

One can try to interpret the automatically learned features from the viewpoints of psychology, anthropology, sociology, culture, ethics. Each movie can tell something about groups of people who rate it: can imply capability of feeling emotions, can tell about personal value systems, moral codes, worldviews, ideological beliefs. The most significant features mark out traits of many users, indicated by many movies. We can surmise that the features depend on gender (features 2+ and 4- likely indicate female, and 4+ male), young age (6+), character traits like sensitivity, neuroticism (feature 1-), or life attitudes like conformism, acceptance, orderliness (feature 1+). We can trace the meaning of features to capability of feeling emotions, like fear or anxiety (feature 5+), or hormone levels heavily influencing emotions, such as high testosterone level (features 4+, 5-) or dopamine (2-, 6+). We could also search for interpretations of the automatically learned information by looking at directions other than the unit vectors in the space of the most meaningful SVD features.

The above interpretations of features, although supported by the observed data, are of course only guesswork, and would require confirmation by conducting appropriate surveys by a trained psychologist.

We can spot in the above examination some connection to movie story types. In the Hollywood Stories dataset, released in 2011, in addition to the standard genres, movies were annotated with the following 22 story types: comedy, love, monster force, quest, rivalry, discovery, pursuit, revenge, transformation, maturation, rescue, escape, the riddle, journey and return, underdog, sacrifice, temptation, fish out of water, metamorphosis, tragedy, wretched excess, rags to riches.

Prediction in the opposite direction is possible – to predict the IMDb tags using the SVD features as predictors, for example, using the logistic regression. Such a prediction can be useful for identifying missing tags, recommending tags in a folksonomy tagging process, or discovering rotations of the feature space – new features most understandable for users. Table 20 shows 50 tags most accurately predicted by the 32 SVD features. Accuracy was measured by the deviance ratio in the logistic regression. The tags tested were a subset of all IMDb tags with support at least 50 among the 2000 most popular movies in the Netflix database.

Table 20: 50 tags best predicted by the SVD features.

chick-flick, crude-humor, spacecraft, teen-movie, future, spoof, warrior, neo-noir, epic, sequel, monster, magic, outer-space, blockbuster, famous-score, psycho-thriller, secret-agent, gothic, gore, based-on-tv-series, robot, sword-fight, alien, surrealism, teen-angst, supernatural-power, battle, blood-spatter, part-of-trilogy, serial-killer, organized-crime, tough-guy, coming-of-age, spy, sword, female-protagonist, impalement, drug-dealing, cult-favorite, shot-in-the-forehead, famous-line, exploding-body, good-versus-evil, castle, time-travel, no-opening-credits, martial-arts, shot-to-death, honor, monkey

Table 21 shows 50 tags (with support at least 50) worst predicted by the SVD features.

Inaccurate prediction indicates that the meaning of those tags is distant from the meaning of the features important for prediction. Those tags are likely needless for predicting ratings or related purposes, and are candidates for removal from the set of tags.

Table 21: 50 tags worst predicted by the SVD features.

title-spoken-by-character, wheelchair, mistaken-identity, dog, bridge, church, funeral, cat, coffee, bus, beach, death-of-friend, drunkenness, reporter, brother-brother-relationship, hospital, airplane, running, father-son-relationship, mirror, one-word-title, california, arrest, friend, christmas, library, interview, jail, punctuation-in-title, birthday-party, kitchen, father-daughter-relationship, fight, train, diner, rivalry, bar, nurse, prologue, bicycle, book, boy, truck, secret, remake, snow, bird, basketball, police-car, manhattan-new-york-city

Now we will look closer at what the SVD features tell us about the standard move genre classification. Going back to explaining SVD features by metadata, table 22 is similar to the previous table 19, but here predictors were only genres. The 6 predictors for each SVD feature were chosen from the 21 standard genres (genre classification by IMDb), as earlier, using greedy feature selection in ridge regression.

Table 22: First six SVD features labeled by standard genres.

Feature	Variance Explained	positive coefficient	negative coefficient
F1	20%	Action, Family, Romance, Sport, Thriller	Drama
F2	33%	Drama, Family, Romance	Action, Comedy, Horror
F3	43%	Drama, Thriller	Action, Animation, Comedy, Family
F4	9%	Action, Crime, Sport, War, Western	Romance
F5	20%	Comedy, Family, Horror	Action, Adventure, History
F6	25%	Action, Family, Sci-Fi, War, Western	Comedy

Some of the six most meaningful SVD features are less well explained than the others by the standard genre labels. This suggests that the standard set of genres needs to be augmented by creating new meaningful genres (bearing in mind, that binary 0-1 genres have limited power of expression, comparing to the SVD features, which can take positive or negative values on a continuous scale). For example, only 9% of variance of the fourth feature was explained by existing genres – the fourth feature was interpreted earlier (table 18) as "violence, aggression, men's world, patriarchy vs. independent women, feminism, matriarchy", or shorter, as "Testosterone vs. Feminism". Based on these guessed meanings, one or two new binary genres could be created to explain better the fourth feature. Features 1, 5, 6 may also suggest new genres. Features 3 and 2 appear to be sufficiently well described by the standard genres. Other linear combinations (other than the unit vectors) of the top SVD features can also be examined and interpreted, to discover important characteristics of movies, unnamed by the standard taxonomies.

One might wonder whether some of the standard genres are unnecessary and can be removed. To find it out, a reverse procedure to the previous was carried out – predicting the binary genres with the logistic regression, using all 32 SVD features as predictors (similar prediction of genres was carried out in [Sel11] on IMDb matched with the MovieLens dataset). The same method was used as in the previous experiment (tables 20, 21), where SVD features were used to predict IMDb keywords. If we assume that all important information about movies is included in the SVD features, the relevant genres should

be predicted well by the SVD features. The genres least accurately predicted by logistic regression, as measured by the ratio of deviance vs. null deviance, were: Adventure, Biography, Crime, Drama, Music, Mystery. Some of these genres are less well predicted, because they are very common, and thus have more "fuzzy" meaning (for example, Drama appears in 1012 movies among the 2000 most popular used in this study). On the basis of the results, in my judgement the candidates to remove are three of the less common genres: Biography, Music (leaving Musical), and Mystery.

To complement the examination of the standard genres, table 23 lays out the average values of first six SVD features for movies (among the 2000 most popular) having the given genre tag. Because the columns 4 and 6 contain only a few, and small negative values, it suggests that the standard genre labelling cannot express the SVD genres 4- (the presumed meaning is "independent women, feminism"), and 6- ("youth, growing up, friendship")

Table 23: Average values of features for standard genres.

Genre (IMDb)	Count	F1	F2	F3	F4	F5	F6
Action	459	0.28	-0.21	-0.03	0.13	-0.07	0.17
Adventure	365	0.16	-0.06	-0.14	0.11	-0.04	0.17
Animation	96	-0.00	0.01	-0.31	0.03	0.01	0.05
Biography	94	-0.25	0.26	0.20	0.03	0.07	0.01
Comedy	826	0.07	-0.03	-0.12	0.00	0.12	-0.05
Crime	407	0.02	-0.12	0.12	0.11	0.02	0.02
Documentary	36	-0.43	0.04	0.00	-0.08	0.02	-0.04
Drama	1012	-0.07	0.12	0.13	0.03	0.05	0.01
Family	206	0.25	0.17	-0.26	0.05	0.12	0.11
Fantasy	233	0.11	-0.08	-0.16	-0.01	0.04	0.12
History	71	-0.10	0.17	0.13	0.12	-0.15	0.15
Horror	112	0.10	-0.38	0.02	0.06	0.19	0.09
Music	128	-0.05	0.17	-0.14	0.01	0.17	0.01
Musical	53	-0.02	0.31	-0.26	0.08	0.16	0.07
Mystery	244	-0.01	-0.07	0.13	0.02	0.01	0.10
Romance	541	0.10	0.19	-0.01	-0.05	0.10	-0.04
Sci-Fi	201	0.11	-0.23	-0.08	0.06	-0.02	0.24
Sport	86	0.17	0.01	0.01	0.19	0.04	-0.08
Thriller	596	0.15	-0.15	0.13	0.09	0.01	0.10
War	75	-0.08	0.10	0.20	0.19	-0.14	0.16
Western	49	0.02	0.01	-0.03	0.26	-0.05	0.19

Table 24 shows the 20 most correlated pairs of genres, and the 20 least correlated pairs. Visible large positive and negative correlations suggest possible inefficiencies and redundancy in the standard set of genres (but not necessarily – correlated directions can have the same expressive power as uncorrelated ones).

One pattern noticed in the feature values is that the average taste changes with popularity. The global average of normalized vectors of item features is (-0.155 -0.032 -0.172 -0.029 0.239 0.212 ...), and the average for the most popular 200 movies is (0.19 0.013 0.057 0.0077 -0.18 -0.11 ...). Various interpretations of the observed pattern are possible: one is that the pattern represents the notion of popular taste. One may wonder here, if the relationship is causal – whether a produced movie has a better chance to become popular when it has features 1+ (idealization), 5- (heroism) and 6- (growing up), and the features to avoid are 1- (realism), 3- (fairy tale), 5+ (innocence) and 6+ (journey). Another interpretation is that the pattern is a result of different inaccuracies in the regularized SVD algorithm appearing for groups of items with largely different amount of ratings, and the automatically learned hidden genres may correct the prediction for those inaccuracies.

Table 24: Largest correlations between average genre vectors.

Genre 1	Genre 2	Cor.
Music	Musical	0.93
History	War	0.92
Biography	Drama	0.86
Crime	Thriller	0.79
Adventure	Fantasy	0.78
Adventure	Sci-Fi	0.76
Action	Thriller	0.76
Action	Adventure	0.75
Action	Sci-Fi	0.75
Fantasy	Sci-Fi	0.73
Mystery	Thriller	0.73
Family	Musical	0.72
Animation	Family	0.72
Family	Fantasy	0.69
Animation	Musical	0.69
Biography	History	0.68
Adventure	Family	0.68
Biography	Documentary	0.65
Animation	Fantasy	0.64
Horror	Thriller	0.62

Genre 1	Genre 2	Cor.
Action	Documentary	-0.64
Action	Biography	-0.63
Comedy	History	-0.58
Comedy	War	-0.58
Music	Thriller	-0.55
Crime	Musical	-0.51
Crime	Music	-0.49
Biography	Sci-Fi	-0.48
Adventure	Documentary	-0.48
Action	Drama	-0.47
Adventure	Biography	-0.47
Biography	Comedy	-0.46
Adventure	Drama	-0.46
Documentary	Thriller	-0.46
Biography	Fantasy	-0.45
Musical	Thriller	-0.45
Drama	Fantasy	-0.45
Crime	Family	-0.43
Comedy	Mystery	-0.41
Drama	Sci-Fi	-0.41

More experiments are needed on varied data to decisively explain this observed pattern.

The above analyses relating SVD features and IMDb genres were heuristic, and should be additionally confirmed by directly modelling the influences of genres and keywords on ratings, by building and training an additional model, instead of utilizing only the SVD results. The notion of "genre" should also be rethought, and more precisely defined. The standard genre theory [Cha97] does not give a precise definition of what is a genre, so I assumed here that genres are named subsets of movies, just like another keywords in the IMDb dataset.

Summarizing, I examined the first six features from regularized SVD, using the Netflix data augmented by IMDb keywords and genres. I listed movies with extreme values of the six features, calculated representations of the features with IMDb keywords and genres, proposed an interpretation of the six features as new genres, and also examined the standard set of genres by IMDb, considering, where it should be augmented by new genres, and which standard genres seem unneeded. The subspace of features learned by an SVD-type algorithm can be spanned by different vectors than the unit vectors – we could choose different rotations of the SVD features that would be more "clean", understandable and intuitive for a human. One might wonder, what causes that humans decide on a taxonomy such as "action", "comedy", "horror" as the dimensions of the space to place movies. It is a question entering the domains of psychology and culture, perhaps related to the concept of meme. Understanding the formation mechanism of such taxonomy could result in dimensionality reduction methods with better explainability, which, as user studies show, is a desirable feature in recommender systems. A similar need to examine, improve or create taxonomies with help of dimensionality reduction methods appears also in other domains, like e.g. for factor analysis methods used in psychometrics.

4.3.5 Improved user preferences

As we mentioned earlier, ratings in the Netflix data are not missing completely at random ($p(Indicators) \neq p(Indicators|Ratings_{observed})$) – users tend to watch and rate movies

they like. This pattern can be used to improve the accuracy of collaborative filtering methods.

As noted in [Mar04, Mar09], there are two basic ways of encompassing missing data structure in a model. One is to treat missing data indicators as additional output variables in a generative model $p(Ratings, Indicators|Hidden)$ (the methods in [Mar04, Mar09] use that approach). Another way, used extensively in the Netflix Prize task was to treat missing data indicators as a fixed structure (not generated variables), on which parts of the model are conditioned $p(Ratings|Hidden)p(Hidden|Indicators)$.

The paper [Sal07a] proposed a Conditional RBM model (described in section 4.4.1), which improves learning of hidden user variables in an RBM by conditioning them on the vector of binary indicators. The indicators influence each hidden user variable through a sum of weights $\sum_{j \in J_i} w_{jk}$ corresponding to each observed rating (note that we can use here also the information from the test set [Sal07a] – an idea called transductive learning). The weights w_{jk} are shared by all users. (the idea called transductive learning).

SVD-type algorithms can be similarly enhanced, as hidden user variables in RBM are analogues of hidden user variables in SVD. [Sal07b] proposes a method Constrained PMF, which enhances PMF (alternating MAP estimation in a probabilistic SVD model) by adding a term $|J_i|^{-1} \sum_{j \in J_i} w_{jk}$ to the user preferences u_{ik}. A similar method, SVD++ [Bel07e], combined matrix factorization (regularized SVD) with NSVD1 [Pat07] by adding the term $u_{ik}^{(0)} = |J_i|^{-0.5} \sum_{j \in J_i} w_{jk}$ to u_{ik}. The parameters of SVD++ were learned by minimizing the regularized MSE cost function on the training data using gradient descent.

Let's describe first the NSVD1 ("New SVD 1") and NSVD2 ("New SVD 2") methods [Pat07], which several variants were included in my final ensemble. The methods are based on the idea of exchanging the user parameters u_{ik} to a function of the binary vector indicating which movies were rated. It turns out that even without looking at the user ratings (except estimating the user bias) we are able to explain a large part of user-specific variability of ratings. In NSVD1 a function of MK new parameters w_{jk} is used. The NSVD1 model, called also asymmetric factor model, is following:

$$\hat{r}_{ij} = \mu + c_i + d_j + (|J_i| + 1)^{-0.5} (\sum_{j_2 \in J_i} \mathbf{w}_{j_2})^T \mathbf{v}_j$$

Noticing the correlation between w_{jk} and v_{jk}, the model NSVD2 using v_{jk} in place of w_{jk} was proposed:

$$\hat{r}_{ij} = \mu + c_i + d_j + (\sum_{j_2 \in J_i} \mathbf{v}_{j_2})^T \mathbf{v}_j$$

All parameters of the models NSVD1, NSVD2 were trained with stochastic gradient descent, but with calculating the sums $\sum_{j_2 \in J_i} \mathbf{w}_{j_2}$ for each user only once per iteration. NSVD1 and NSVD2 do not give very accurate predictions, but they combined well with ensembles of other methods (also in prediction ensembles for other datasets, such as music ratings [Jah11a, Jah11b]).

Blending regularized SVD with NSVD1 by linear regression gave better accuracy than using regularized SVD alone. Such a situation is an indication, that it is possible to combine both methods to obtain a single method with accuracy at least as good as the blend of the two methods. And indeed, SVD++ methods [Bel07e, Bel07f, Bel08] were a way to combine SVD with NSVD1 with good outcome. The SVD++ model has the form:

$$\hat{r}_{ij} = \mu + c_i + d_j + (\mathbf{u}_i + |J_i|^{-0.5} \sum_{j_2 \in J_i} \mathbf{w}_{j_2})^T \mathbf{v}_j$$

The initial idea behind NSVD1 and NSVD2 models was to reduce the number of learned parameters (if not counting the use of implicit information about movies selected),

as fewer parameters can lead to reducing overfitting. In NSVD1 and NSVD2 the learned user preferences depend on user ratings only through weights w_{jk}, which are common for all users. It leaves less opportunity to overfit, at the cost of reduced capability to accurately model user preferences (the normalized sum of weights w_{jk} does not even have the expressive power to predict a fixed constant $|J_i|^{-0.5}\sum_{j\in J_i} w_j \approx 1$, and introduces noise – on a side note, similar limitations imposed by the structure of summations exist in multitask ridge regression used to calculate user preferences in the regularized SVD).

Moving to the topic of improving regularized SVD, in my experiments I tried out several ways of improving the SVD model by predicting the mean preferences of each user. Most of the tried out methods did not improve RMSE significantly, for example, using for prediction the K-means clustering of user features. The only methods that improved RMSE were those using missing data in similar way to SVD++ and NSVD1. Predicting the means was used both in the neural-networks-like models and in the Bayesian SVD models. In the Bayesian models the predicted means $u_{ik}^{(0)} = |J_i|^{-0.5}\sum_{j\in J_i} w_{jk}$ were used in the prior distribution of u_{ik} with the remaining regularization parameters chosen by automatic parameter tuning (it is also possible to estimate the prior variance of u_{ik}, different for each user, instead of tuning regularization). In neural-networks-like regularized SVD, the term $u_{ik}^{(0)}$ was added to the learned user preference parameter: $u_{ik}+u_{ik}^{(0)}$. The difference between my implementations and the formulation of SVD++ [Bel07e] was that I repeatedly optimized the parameters w_{jk} to minimize $\sum_{ij\in Tr}(u_{ik} - |J_i|^{-0.5}\sum_{j\in J_i} w_{jk})^2$, instead of adding the $u_{ik}^{(0)}$ term to the global cost function $\sum_{ij\in Tr}(r_{ij} - \hat{r}_{ij})^2$. The weights w_{jk} were optimized by gradient descent, with varied methods used to optimize the remaining parameters u_{ik}, v_{jk}, c_i, d_j.

Table 25: Methods with modelling user preferences using the missing data structure. Experimental results.

Method	K	RMSE_{15}	RMSE_{quiz}
Conditional RBM [Sal07a]	40		($\text{RMSE}_{probe} \approx 0.9070$)
Conditional RBM [Bel07c]	200		0.9029
NSVD1 [Bel07c]	40		0.9260
NSVD1 [Tak08c]	80		0.9344
NSVD1 [Tos09]	800		0.9213
NSVD2 [Bel07c]	40		0.9383
SIGMOID3 [Bel07c]	30		0.9194
Constrained PMF [Sal07b]	60		0.9016
SVD++ [Bel07e]	60		0.8970
SVD++ [Kor08]	100		0.8924
Asymmetric SVD [Kor08]	100		0.9013
HYBRID1 [Tak08c]	400		0.8871
MFW31-00milestone6-100 [Pio09]	100		0.9235
NSVD1	40	0.9328	($\sim$ 0.9360)
NSVD1+BASIC [Pat07]	40	0.9312	($\sim$ 0.9340)
NSVD2	5	0.9624	($\sim$ 0.9655)
NSVD2+BASIC [Pat07]	5	0.9590	($\sim$ 0.9620)
NSVD2 w/o qual	5	0.9633	($\sim$ 0.9660)
NSVD1B	50	0.9464	($\sim$ 0.9490)
QNSVD1	85	0.9679	($\sim$ 0.9710)
NSVD1R	40	0.9401	($\sim$ 0.9430)

In several implemented methods I used splitting users on 5 groups according to their support, and learning one additional constant per group, by which the term $u_{ik}^{(0)}$ was multiplied. The learned five constants deviated from 1.0 (see the description of SVDM method in section 4.3.6), which suggests that it is possible to find a better normalization

than $|J_i|^{-0.5}$, but the methods with learning the five constants did not improve accuracy significantly.

[Pio09] proposed a modification of SVD++ called "milestone", which gave accuracy improvement by splitting the parameters w_{jk} in the sum $\sum_{j \in J_i} w_{jk}$ into a convex combination of two sets of parameters: $\sum_{(j,t) \in J_i} a_{if_{it}} w_{jk} + (1 - a_{if_{it}}) w'_{jk}$, the parameters being weighted by frequency-dependent per-user variables $a_{if_{it}}$ (frequency f_{it} is the number of movies rated by user i on a given day t).

Table 25 lists the methods that extend the user preferences with the use of binary missing data information. The methods NSVD1 and NSVD2 in the table were described in [Pat07]. "NSVD2 w/o qual" does not use the information from qualifying.txt about movies selected for rating. NSVD1B is NSVD1 with weights $e_i = 1$. NSVD1R is an NSVD1 version that uses residuals of ratings instead of binary information on users selected. QNSVD1 is NSVD1 with weights $e_i = 1$, and using only the information on movies selected from probe and qualifying set (without the information on movies selected from the training set). All the above methods were parts of the ensemble in [Pat07], and also contributed to the final ensemble of this work (chapter 5 "Experimental results").

Many of the most accurate methods in my ensemble were based on SVD++ variants, enhanced by time effects, and combined with K-NN or postprocessed by KRR. These SVD++ variants were listed in other sections: 4.3.6 "Time effects", 4.5.3 "Kernel methods", and 4.8.1 "Preprocessing and postprocessing".

Figure 32: Relationship between v_{ik} and w_{jk}

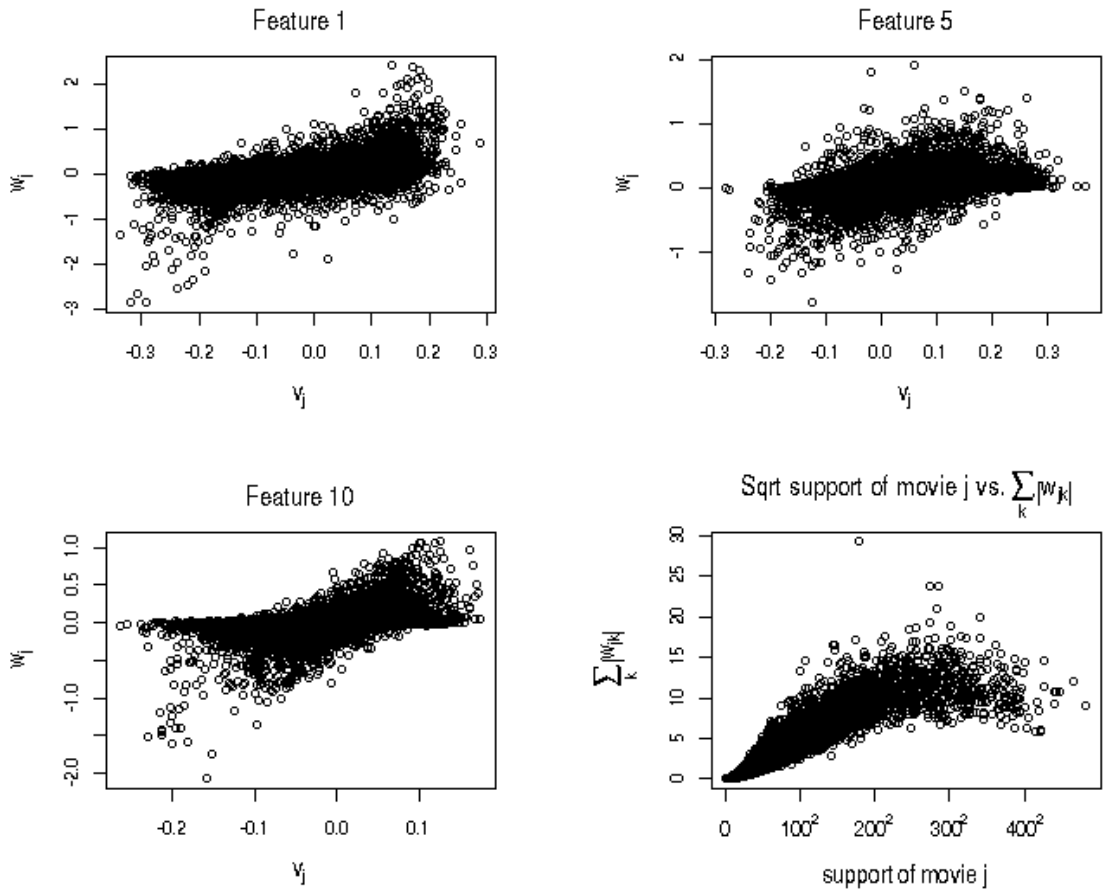


Figure 32 shows three scatter plots demonstrating the relationship between the parameters v_{jk} and w_{jk} in an SVD++ variant, for features 1, 5, 10, and a scatter plot of

$\sum_{j \in J_i} |w_{jk}|$ plotted against the square root of movie support. Further examination showed, that the source of the pattern observed on the first three plots is that the correlation between v_{jk} and w_{jk} increases with the increasing amount of user ratings.

A "flipped", movie-oriented model was tried out in [Tos08b, Tos09], where weights w_{ik} were summed to model movie features $|I_j|^{-0.5} \sum_{i \in I_j} w_{ik}$. Flipped versions of SVD++ were very accurate on a task of separating high ratings from unrated items in music ratings data [Jah11b].

An open question left is how to properly use the missing data information to improve multitask Kernel Ridge Regression (Gaussian Processes) approach (see section 4.5.3). There were several attempts to use PCA of the binary matrix of indicators, for example, BSRM/F [Zhu08].

The described way of using missing data turned out to improve accuracy of models in the task of predicting ratings from held-out training data, but in situations when we need to predict ratings for non-user-selected items, as is the case of calculating personalized recommendations, it is likely that a more sophisticated way of using missing data will be needed.

4.3.6 Time effects

The assumption behind SVD-type algorithms is persistence of item properties, and persistence of user taste. This assumption is largely satisfied on typical datasets with movie ratings, which is the reason why collaborative filtering algorithms and recommender systems work well, but it is not satisfied perfectly – the model of data changes over time. We can notice both short-term deviations from the static model, as one-day, or one-user-session, and we can notice the change of variables, trends, patterns over longer periods of time. Time effects can be divided into user-side, on user bias and user preferences, and movie-side, on movie biases and movie features. On the Netflix task, including the user-side time effects in the models gave a greater improvement of accuracy than including the movie-side effects. This section lists time effects that improved accuracy of the regularized SVD the most (analogous time effects improve also other families of collaborative filtering algorithms), and briefly describes several most accurate time-aware models developed by different authors.

The most important time effects on the user variables were: modelling variability of user bias over time, modelling variability of user preferences, and making use of daily rating frequency, which enters in significant interactions with other variables.

The largest time effect in the Netflix data is the single-day correction of the user bias. It can be modelled with independent variables, one per each day (proposed in [Bel08] as part of the model, and in [Pot08] as a global effect, but also was part of the unpublished model Bayesian PCA [Tom07]). The average number of additional variables per user is about 40 [Kor09b]. The one-day bias was usually realized by direct regularized point estimation (or optimizing with gradient descent):

$$c_{it} = \frac{\sum_{j \in J_{it}} \hat{y}_{ij}}{|J_{it}| + \lambda} \tag{28}$$

where $\hat{y}_{ij}$ is the residual of the remaining part of the model. In [Kor09b] additional per-day scaling parameter was proposed, shared by several effects:

$$c_{it} = s_{it} \frac{\sum_{j \in J_{it}} \hat{y}_{ij}}{|J_{it}| + \lambda}$$

To model longer correlations than one-day, we can split the parameter into longer periods of time, using less parameters (a method called binning), or using methods such as

exponential smoothing or convolving the user bias with time-dependent kernels [Tos08b]. In some of the implemented methods I used a one-directional exponential moving average of residuals:

$$c_{it_2} = \frac{\sum_{j,t\in J_i:t\leq t_2} \exp(C|t_2 - t|)\hat{y}_{ij}}{\sum_{j,t\in J_i:t\leq t_2} exp(C|t_2 - t|) + \lambda}$$

or a bidirectional moving average:

$$c_{it_2} = \frac{\sum_{j,t\in J_i} \exp(C|t_2 - t|)\hat{y}_{ij}}{\sum_{j,t\in J_i} exp(C|t_2 - t|) + \lambda}$$

It is also possible to model the fluctuation in time with Ohrstein-Uhlenbeck process (idea by [Tom07]), or its discrete version, AR process. The Bayesian PCA model [Tom07] included a small amount of continuity between variables on adjacent days.

In addition to correlated nearby residuals of the model outputs, a user bias can be the subject of a trend – it can increase or decrease along a time-dependent curve. In [Bel07b] the following per-user global effect was proposed:

$$c_{it} = \alpha_i \ (t - t_i^{(0)})^{0.5} \tag{29}$$

where $t_i^{(0)}$ is the date of the first rating of user i, and α_i is the fitted user parameter. An analogous per-movie global effect was also used. [Bel08] included the trend term (29) as part of the model, with the exponent 0.4 instead of 0.5.

Because the dataset contains over 480 thousands users, it is possible to try automatic learning of repeating trends in the users' biases. For example, similarly to using user-item factorization in regularized SVD, we can use factorization to express a trend in user's ratings with a linear combination of a small number of automatically learned trends $z_{\tau k}$ ([Pio09, Xia09]):

$$c_{it} = \mathbf{x}_i^T \mathbf{z}_\tau \tag{30}$$

where $\tau = t - t_i^{(0)}$. A similar method modifying the product of the user preference u_{ik} and the movie feature v_{jk} by a factor varying in time $z_{\tau k}$, was used in [Tak07a, Xia09, Xio09]:

$$c_{ijt} = \sum_{k=1} u_{ik} v_{jk} z_{\tau k}$$

We should note that the above idea of capturing trends using factorization does not work so well, and other time effects give much larger accuracy improvements.

There was also a proposal to add a global effect – a bias b_t for each day t, common for all users. As seen on the plot 17 in section 3.5, the average of ratings visibly changes over time. It turns out, that the global per-day variations are rather well explained by the remaining variables of the model, so the day bias b_t did not give a significant accuracy improvement, and was not part of the most accurate models [Kor09b, Tos09, Pio09].

Similar methods to the used for biases can be used to model variability of user preferences $\mathbf{u}_{ik}(t)^T \mathbf{v}_{jk}$ over time. The following formula was proposed in [Bel08, Kor09b] to model user preferences:

$$u_{ik}(t) = u_{ik} + \alpha_{ik} dev_i(t) + u_{ik,t}$$

where $dev_i(t) = sign(t - t_i^{(0)})|t - t_i^{(0)}|^{0.5}$. Hence two time effects are added to the ordinary user preferences u_{ik}: a variable α_{ik} capturing the trend, and a variable $u_{ik,t}$ capturing the one-day correlation of residuals. Adding the term $u_{ik,t}$ resulted, on average, in about $40,000$ additional variables per user in the model. Modelling time correlations is easier in the dual method to the ridge regression, KRR (see section 4.5.3 "Kernel methods"), where capturing correlations in time is reduced to modifying the covariance matrix between residuals of item ratings.

In addition to user time effects, some accuracy improvement resulted from modelling per-movie time effects. Simple binning was most commonly used for that purpose:

$$c_{j,bin} = \frac{\sum_{i \in I_{j,bin}} \hat{y}_{ij}}{|I_{j,bin}| + \lambda}$$

where $I_{j,bin}$ are users who rated movie j on dates $t \in bin$. In the paper [Kor09b] dates were divided into 30 equal-sized bins, each spanning about 10 weeks. Also, factorization of movie bias was proposed, expressing the change of bias in time with a linear combination of jointly learned trends:

$$d_{jt} = \mathbf{x}_j^T \mathbf{z}_\omega \tag{31}$$

where $\omega = t - t_j^{(0)}$ ($t_j^{(0)}$ is the date of the first rating of movie j).

A group of time effects, which inclusion improved accuracy, were frequency based effects [Pio09], that is effects including the frequency component $F_{it} = |J_{it}|$ (the number of ratings given by user i on one day). A plausible explanation [Pio09, Kor09b] of why the frequency effects exist in the Netflix data, is that a user rates movies in a different way, when he watched the movie recently, and differently when he watched the movie long time ago. The time elapsed since viewing the movie is not observed, but indicated indirectly by the frequency – if a user gives few ratings on one day, it indicates that the user likely watched the rated movies recently. Interactions that turned out to be useful were interactions of frequency with item-side variables [Pio09, Kor09b, Tos09]: movie biases c_j, movie features v_j, NSVD-part variables w_j, and global neighborhood weights z_j. In BK4 models [Pio09] and SBRAMF-* [Tos09] the value $f_{it} = F_{it}$ was used directly to create a separate set of new variables for each possible value of frequency, for example, the movie bias d_j becomes $d_{j,f_{jt}}$. [Kor09b] used the rounded logarithm of frequency $f_{it} = \lfloor \log |F_{it}| \rfloor$ to index $d_{j,f_{jt}}$ terms, and in [Pio09] frequency was split into intervals of different sizes ($8 - 36$ intervals).

I will list now several very accurate variants of time-dependent regularized SVD proposed by the winners of the Netflix Grand Prize [Bel08, Tos08b, Kor09b, Tos09, Pio09] and by other teams [Xia09]. The parameters in the following models were trained by gradient descent minimization of a regularized cost function SSE (sum of squared error) on the training set.

TimeSVD++ [Bel08, Kor09b] adds time effects to user biases, movie biases, and user preferences (in the notation of this work);:

$$\hat{r}_{ui}(t) = \mu + c_i + c_i' dev_u(t) + c_{it} + d_j + d_{j,Bin(t)} + (\mathbf{u}_i + \mathbf{u}' dev_u(t) + \mathbf{u}_{it} + |J_i|^{-0.5} \sum_{j_2 \in J_i} \mathbf{w}_{j_2})^T \mathbf{v}_j$$

where

$$dev_u(t) = sign(t - t_i^{(0)}) |t - t_i^{(0)}|^{0.4}$$

where $t_i^{(0)}$ is the date of the first rating given by user i.

The method PQ2 [Kor09b] adds to TimeSVD++ [Bel08] per-user scaling, and, following [Pio09], adds additional movie-side parameters for different levels of the user daily frequency:

$$\hat{r}_{ui}(t) = \mu + c_i + c_i' dev_u(t) + c_{it} + (d_j + d_{j,Bin(t)})(s_i + s_{it}) + d_{j,f_{it}} +$$

$$+ (\mathbf{u}_i + \mathbf{u}' dev_u(t) + \mathbf{u}_{it} + |J_i|^{-0.5} \sum_{j_2 \in J_i} \mathbf{w}_{j_2})^T (\mathbf{v}_j + \mathbf{v}_{j,f_{it}})$$

The authors note in [Kor09b], that in contrast to frequency-aware movie biases, frequency-aware movie features did not improve the model by much.

The method SBRAMF-UTB-UTF-MTF-ATF-MFF-AFF [Tos09], listed in table 26 as SBRAMF-*, is a variant of TimeSVD++ [Bel08, Kor09b] extended by frequency-based

movie features and frequency-based asymmetric features. The method includes extended regularization of parameters from the model SBRISMF [Tak08b] (see section 4.3.2). The model has the form:

$$\hat{r}_{ui}(t) = c_i + c_{it} + d_j + d_{j,Bin(t)} +$$
$$+ \left(\mathbf{u}_i + \mathbf{u}_{it} + |J_i|^{-0.5} \sum_{j_2 \in J_i} (\mathbf{w}_{j_2} + \mathbf{w}_{j_2,Bin(t)} + \mathbf{w}_{j_2,f_{it}})\right)^T (\mathbf{v}_j + \mathbf{v}_{j,Bin(t)} + \mathbf{v}_{j,f_{it}})$$

The method BK4 [Pio09] is TimeSVD++ [Bel08] integrated with the global neighborhood model [Kor08], extended by frequencies, and factorization of time-dependent user biases (30), movie biases (31), and per-user scaling. Because the BK4 model contains a global neighborhood component, it is listed in section 4.8.2 "Integrated models". The BK4 method postprocessed by K-NN [Pio09] (see section 4.8.1 "Preprocessing and postprocessing") has the best RMSE accuracy among the published methods.

Table 26 lists a method TimeSVD++ + STE [Xia09], which uses factorized time-dependent user biases (31), movie biases (30), and user preferences, global time bias, and several additional time effects: year and month effect, and effects named loyalty, activity and popularity [Xia09].

The bottom of the table 26 lists the time dependent methods implemented by me. SVDM is a version of regularized SVD with a regularization form inspired by the learning equations of VB SVD, extended by using user day bias c_{it}, estimated by shrinked average daily residuals as in the equation 28, with $\lambda = 20$, but in a leave-one-out version, skipping the current residual $\hat{y}'_{ij}$ during estimation. Predictions in SVDM have the form: $\hat{y}_{ij} = \mu + c_i + d_j + c'_{it}(j) + \mathbf{u}_i^T \mathbf{v}_j$. User preferences and movie features are estimated one parameter at a time by a jump to the marginal minimum of the regularized cost function:

$$u_{ik} = \frac{\sum_{j \in J_i} v_{jk} \hat{y}_{ij}^{(k)} + \overline{u}_{ik} \lambda_1 / \tau_k^2}{\sum_{j \in J_i} v_{jk}^2 + \lambda_1 / \tau_k^2 + \lambda_2 |J_i|} \qquad v_{jk} = \frac{\sum_{i \in I_j} u_{ik} \hat{y}_{ij}^{(k)}}{\sum_{i \in I_j} u_{ik}^2 + \lambda_3 + \lambda_4 |I_j|}$$

where the mean of the prior user preference $\overline{u}_{ik}$ is estimated by NSVD/SVD++ (section 4.3.5 "Improved user preferences"), with the NSVD term rescaled in five groups of users with different level of support $C(|J_i|)|J_i|^{-0.5} \sum_{j \in J_i} w_{jk}$ by the following constants found by automatic parameter tuning: $C(|J_i|) \in \{0.7, 1.66, 1.88, 1.45, 0.99\}$, and with analogous five distinct learning rates. The parameter τ_k^2 approximates the variance of prior distribution $Var\ u_{\cdot k}$ of the given feature k. The current residuals $\hat{y}_{ij}^{(k)}$ are ratings with all global effects and features subtracted, except the current feature k: $\hat{y}_{ij}^{(k)} = r_{ij} - \hat{y}_{ij} + u_{ik} v_{jk}$. The number of features K was set to 30. The regularization parameters were chosen automatically by the Praxis procedure: $\lambda_1 = 0.77$, $\lambda_2 = 0.022$, $\lambda_3 = 3.14$, $\lambda_4 = 0.005$.

SVDMT is SVDM extended by time-dependent user and movie biases – biases c_i and d_j were split into 32 bins with a length of 70 days. SVDMT postprocessed by KNN (details in sections 4.8.1 and 4.5.1), with the training of SVDMT and KNN repeated twice on residuals of each other, gave the predictor SVDMT2KNN, the most accurate method developed in this work with $RMSE_{15} = 0.8819$.

Table 26 compares RMSE of different methods. The time-aware methods that additionally user neighborhood models were moved to the table 39 in section 4.8.2 "Integrated models".

Summarizing, using time effects similar to the listed ones was one of the keys to pass the barrier of 10% improvement of RMSE over the reference algorithm, and winning the Netflix Prize competition [Bel08, Tos08b, Kor09b, Tos09, Pio09].

It turned out that the time dependent user bias was the effect, including which improves RMSE the most, but in a real recommender system, at the moment of calculating a recommendation list predictions for all movies have the same user bias. We can conclude

Table 26: Models with time effects.

Method	Learning	Grouping	K	RMSE_{15}	RMSE_{quiz}
Bayesian PCA [Tom07]	single	MCMC	60		0.8805
TimeSVD++ [Bel08]	multi	gradient	200		0.8806
TimeSVD++ [Kor09a]	multi	gradient	200		0.8799
PQ2 [Kor09b]	multi	gradient	200		0.8777
Integrated Model [Tos09]	multi	gradient	150		0.8806
SBRAMF-* [Tos09]	multi	gradient	150		0.8788
mfw31-60-10-120-m [Pio09]	multi	gradient	150		0.8883
TimeSVD + STE [Xia09]	multi	gradient	100		0.9027
BPTF [Xio09]	multi	gradient	100		0.9044
SVDM	single	MAP VB-like	30	0.8919	($\sim$ 0.8950)
SVDMT	single	MAP VB-like	30	0.8909	($\sim$ 0.8940)

that the time effects on the user bias are less important than it is indicated by the RMSE criterion.

4.3.7 Matrix norm regularization

In the cost-function-based approach to regularized SVD, instead of regularizing the parameters u_{ik}, v_{jk} (entries in matrices U and V), a close approach is to regularize directly the approximation matrix $X = UV^T$ with a matrix norm. Minimizing the squared loss with matrix norm regularization is of interest to us, because it provides a simplified view of certain variants of the regularized SVD (although the developed algorithms for matrix norm regularization are SVD-based anyway). An important case is that a constant regularization of U and V (non-optimal for our data) is equivalent to regularizing $X = UV^T$ with the trace norm.

The trace norm of a matrix X, called also the nuclear norm, or the Schatten 1-norm, is the sum of the singular values of a matrix: $||X||_\Sigma = \sum_{k=1}^{M} \sigma_k$. Minimizing the trace norm was proposed as a heuristic in place of minimizing the matrix rank [Faz01, Faz03]. Trace norm regularization was used in Maximum Margin Matrix Factorization (MMMF) [Sre04, Sre05, Ren05] for collaborative filtering (but with the hinge loss, unlike the squared loss used here). MMMF was tried out on the Netflix Prize data in [Wu07], where it contributed to the ensemble of 16 differing matrix factorizations. Various algorithms for matrix norm regularization with different loss function choices were proposed in [Cai08, Ma08, Jag10]. Minimizing the trace norm was also used in theoretical research on matrix completion [Can09, Sal10].

As noted in [Faz01, Sre04]:

$$\min_{X=UV^T} \frac{1}{2}(||U||_F^2 + ||V||_F^2) = ||X||_\Sigma$$

This implies, that the regularized SVD with a constant regularization parameter λ, and with relaxing the constraint on the rank of the approximation matrix UV^T, is equivalent to trace norm regularization (S is a 0-1 matrix indicating the observed matrix elements):

$$\min_{U,V} \; ||Y_S - (UV^T)_S||_2^2 + \lambda||U||_2^2 + \lambda||V||_2^2 = \min_{X} \; ||Y_S - X_S||_2^2 + \lambda||X||_\Sigma \qquad (32)$$

If the matrix Y is fully observed ($S = \mathbf{1}_N \mathbf{1}_M^T$), then the optimization of X in the cost function with the trace norm regularization is a convex problem, in contrast to optimization of U and V with the Frobenius norm regularization of U and V (but for the optimization of U and V all local minima are global [Sre04]). The solution $\hat{X}$ is a global minimum of

(32) if and only if [Cai08]:

$$Y - \hat{X} \in \lambda\delta||X||_\Sigma \tag{33}$$

where $\delta||X||_\Sigma$ is the set of subgradients of the trace norm:

$$\delta||X||_\Sigma = \{UV^T + W : W \in R^{N\times M}, U^TW = 0, WV = 0, ||W||_2 \leq 1\} \tag{34}$$

One can verify by substitution, that (33), (34) is satisfied by a shrinked result of a dense SVD of the matrix Y:

$$\hat{Y} = \sum_{k=1}^{K} \max(0, \gamma_k - \lambda)\mathbf{u}_k\mathbf{v}_k^T \tag{35}$$

where $\mathbf{u}_k$ is the k-th column of U, $\mathbf{v}_k$ is the k-th column of V, and γ_k is the k-th singular value in the SVD of the matrix $Y = U\Sigma V^T$.

The above solution with shrinked singular values of a standard, linear algebra SVD works for a fully observed matrix Y. If relatively few data values are missing (and the data is missing uniformly), EM-like data imputation algorithm can be used. The algorithm Singular Value Thresholding [Cai08, Liu09] repeatedly iterates between calculating the current approximation X of the matrix Y_S of observed values, and filling the missing values with the result of the shrinked SVD (35). If, in turn, a large part of the matrix is missing, data imputation leads to inaccuracy, but if we ignore the missing data, U and V can be learned, for example, with gradient descent [Ren05, DeC06].

I tested a modification of the SVT [Cai08] algorithm on a subset of the Netflix Prize dataset with less than 50% data missing. The modification was inspired by the form of the solution of dense Variational Bayesian SVD [Nak09, Nak10a, Nak10b, Nak11a, Nak11b] (let's remind that sparse VB SVD are among the most accurate methods of matrix factorization for the Netflix Prize task).

In [Nak09], for the probabilistic model of matrix factorization: $y_{ij} \sim N(\mathbf{u}_i^T\mathbf{v}_j, \sigma^2)$, $u_{ik}^{a-priori} \sim N(0, \tau_u^2)$, $v_{jk}^{a-priori} \sim N(0, \tau_v^2)$ the analytic solution of the Variational Bayesian approximation is calculated for a case when all data is observed:

$$\hat{Y} = \sum_{k=1}^{K} \hat{\gamma}_k\mathbf{u}_k\mathbf{v}_k^T \qquad\qquad \hat{\gamma}_k = \max(0, \gamma_k - \max(M, N)\sigma^2\frac{1}{\gamma_k} - \Delta_k) \tag{36}$$

where U, V, γ_k come from the standard linear algebra SVD of training data. [Nak09] gives also bounds on Δ, when $N \neq M$, and the exact solution, when $N = M$. The form of the analytic solution (36) of VB suggests [Nak09], that in the assumed probabilistic SVD model, with a fully observed matrix, VB inference is equivalent to combining trace norm regularization, positive-part James-Stein (PJS) shrinkage, and Frobenius norm regularization.

The modification of SVT in my experiment, comparing to [Cai08], was changing the shrinkage of singular values to $C_1\gamma_k - C_2/\gamma_k - C_3$, with adaptively choosing the three parameters C_1, C_2, C_3 in the each interaction, using linear regression on the test set (one iteration fills missing data with the result of modified SVT from the previous iterations, and then calculates the new shrinked SVD on the augmented dense matrix). The form of shrinking $C_1\gamma_k - C_2/\gamma_k - C_3$ was chosen after trying several plausible forms. The modified SVT was run on a small subset of the Netflix Prize data, which is 50% dense (56% dense together with the test set) – how that dataset was selected, it was described in section 3.3. Table 27 shows the result of the modified SVT. In chapter 5 it is compared with other algorithms trained on the same subset of the Netflix data.

After 10 iterations of the method, the learned weights (rounded): $C1 = 1.023$, $C2 = 2560$, $C3 = 16.6$, turned out to be close to regularization by combining the trace norm and the PJS estimator, which corresponds [Nak09] to VB SVD. The resulting weights

Table 27: Models with matrix norm regularization. Experimental results.

Method	K	Iterations	RMSE_{small}
SVT	80	10	0.7449

are an additional argument (to the all experiments of many teams working on the Netflix data), that the considered model of probabilistic factorization is close to the unknown optimal model.

A question remains whether to limit the approximation to a function of singular vectors from the linear algebra SVD – whether it is not worth to go beyond the matrix norms that are functions of singular values of the approximated matrix. One attempt was to use matrix completion with max-norm regularization [Cho11].

The modified SVT described above was tested on a subset of the Netflix Prize dataset that is 50% sparse. For the entire Netflix dataset, which is about 99.8% sparse, more appropriate are algorithms like regularized SVD, which do not impute missing values. Also, constant regularization λ is inferior to the regularization amount increasing linearly with the number of observations (the number of user ratings to regularize U, and the number of movie ratings to regularize V). The rescaled regularization has the following form:

$$||D_U X D_V||_\Sigma = min_{X=UV^T} \frac{1}{2}(||D_U U||_F^2 + ||D_V V||_F^2)$$

where D_U is a diagonal matrix containing $\sqrt{\lambda_{U1} + \lambda_{U2}|J_i|}$ on the diagonal, and D_V is a diagonal matrix containing $\sqrt{\lambda_{U1} + \lambda_{U2}|I_j|}$ on the diagonal ($|J_i|$ is the number of ratings given by user i, and $|I_j|$ the number of ratings given to movie j). Using a linear regularization can be justified not only experimentally, or by the form of approximate Bayesian SVD – as shown in [Sal10], the linear amount of regularization follows from considering the matrix norm of a matrix composed by disjoint submatrices of different sizes.

In summary, the cost-function-based formulations of the collaborative filtering that use the matrix regularization by a spectral norm, ultimately come down to SVD-type algorithms, and this is an additional justification for studying the algorithms in the form of regularized SVD directly.

An advantage of the SVT algorithms is that they can be easily realized using the standard dense SVD implementations in libraries, avoiding convergence and tuning issues that appear in gradient-descent-based algorithms. Algorithms similar to SVT may be useful when obtaining best possible accuracy is not a priority, on datasets with a small amount of missing data. The drawbacks of SVT are necessity of imputing missing values and the used non-optimal, constant regularization. A natural direction of further development of the matrix norm regularization algorithms is proposing an algorithm with regularization rescaled by constant-linear terms – the form of regularization that worked the best on the Netflix Prize task and similar collaborative filtering tasks.

4.3.8 Other matrix factorization methods

The regularized SVD method, in its basic form, is based on the model $\hat{r}_{ij} = \mu + c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j$. Other variants of matrix factorization were tried out, as generalized matrix factorizations, or non-negatively constraining the variables (by "matrix factorizations" I understand all models containing the product of continuous hidden variables representing user preferences and item features $\mathbf{u}_i^T \mathbf{v}_j$).

For different types of data and different collaborative filtering tasks, different dimensionality reduction methods are appropriate. It is natural to try models containing a

product of hidden variables, but transformed by a function chosen suitably to the data (for example, many variants of matrix factorizations were listed in [Sin08, Sin09, Del08]). The idea of generalized matrix factorization is to change the model $Y = UV^T$ (or a version with biases $Y = C + D + UV^T$) to $y_{ij} = g(\mathbf{u}_i^T \mathbf{v}_j)$ (resp. $y_{ij} = g(c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j)$), where g is a non-linear transformation function (link function). Generalized matrix factorizations were useful in the Netflix Prize task to bound the output of matrix factorization (predicted rating) to the range $1 - 5$. In this work, the only way used of bounding the output was clipping predictions to range 1 to 5 when training and during making predictions (except for two methods, SVDB1 and SVDB2, which used MF with logistic transformation predicting probability that a movie was rated by the user). Some accuracy improvement can be obtained by using a smooth transformation of the outputs, for example, in [Pio09] a shifted sigmoid transformation was used:

$$\sigma(x) = \begin{cases} 1 + \frac{2(\sigma_0 - 1)}{1 + \exp(\frac{-2(x-\sigma_0)}{\sigma_0 - 1})} & \text{if } x < \sigma_0 \\ 2\sigma_0 - 5 - \frac{2(5 - \sigma_0)}{1 + \exp(\frac{-2(x-\sigma_0)}{5-\sigma_0})} & \text{if } x \geq \sigma_0 \end{cases}$$

The OrdRec model [Kor11] improves accuracy of SVD++ by treating the ratings as being on an ordinal scale, with user-specific thresholds. Other ordinal MF models were proposed in [Ste09, Paq10].

Another variation of matrix factorization, non-negative matrix factorization (NNMF) [Lee00, Zha06] is to non-negatively constrain the variables. NNMF contributed to ensembles of several teams in the Netflix Prize [Wu07, Bel07c], and was the most accurate factorization in the ensemble in [Bel07c]. In [Pio09] non-negative matrix factorizations were sometimes used with an inverted rating scale (1 switched with 5). In my experiments I tried imposing non-negativity on one of the variables in the regularized SVD, by modifying the model to $\sum_{k=1}^{K} \exp(u_{ik}) v_{jk}$ or $\sum_{k=1}^{K} u_{ik} \exp(v_{jk})$. Those methods were present in the ensemble for some time, but ultimately were removed, when they stopped improving accuracy.

Smooth convex polyhedron prediction (SMAX) [Tak09b] uses multiple sets of user preferences, and combines the resulting factorizations using a selected smooth maximum function.

In BMFSI with Dirichlet Process Mixtures [Por10a, Por10b], a variant of matrix factorization, user preferences and movie features were augmented with side information (similarly as in [Tak07a]), and several different priors for feature vectors were learned. The prior for each vector was selected in the model by clustering users and movies with Dirichlet Process Mixtures. The clustering gave a small accuracy improvement. A larger improvement was obtained by using the following additional features as a side information: the normalized date of a rating, ratings for the five nearest neighbors (the most similar movies) and the previous two ratings given by the user.

Another accurate variant of MF was Mixed Membership Matrix Factorization (M^3F) [Mac10], in which user and item topics affect the rating through a contextual bias added to the basic matrix factorization model. The user and item topics are realized as a mechanism of mixed membership, that is using a discrete variable with learned user-specific and item-specific Dirichlet priors, similarly as in the LDA algorithm [Ble03], and, for dyadic data, in Bi-LDA [Por08, Por10b]. Two types of contextual bias were proposed in [Mac10]: Topic Indexed Bias (TIB) and Topic Indexed Factor (TIF). Table 28 lists RMSE of the TIB version. TIB augments the basic probabilistic matrix factorization model with additional user biases, one for each user topic, and item biases, one for each item topic.

Nonparametric Bayesian Matrix Completion [Zho10b] is a matrix factorization model modified by drawing, for each column of the matrix, a separate set of binary selectors from a Bernoulli distribution, specyfying which singular values are zeroed.

Matrix factorization models can be generalized to modelling simultaneously multiple relationships [Zha09], for example, a user-item relationship (collaborative filtering) together with a user-user relationship (such as a list of trusted users).

Bayesian Factorization Machines [Fre11] are able to model different kinds of patterns (time effects, implicit information, K-NN, interactions between groups of features, etc.) using binary or real-valued features. Features and combinations of two or more features are weighted by factorized parameters, creating a tensor factorization model, with parameters learned by unblocked Gibbs sampling. The variant BFM(u,i,t) [Fre11] listed in table 28 used the user id, the item id, and the date of rating as features. Feature-Based Matrix Factorization [Che11b] is a restricted version of BFM.

Table 28 summarizes RMSE's of the mentioned above alternative variants of matrix factorizations.

Table 28: Other matrix factorization methods.

Method	Learning	K	RMSE_{quiz}
NNMF [Bel07c]		128	0.9039
NNMF AUF-MseSim [Bel07c]		128	0.8955
BMFSI (DP) [Por10a]	MCMC	45	0.8957
BMFSI (DP,SI) [Por10a]	MCMC	100	0.8875
M^3F-TIB [Mac10]	MCMC	120	0.8934
Ordinal MF [Paq10]	MCMC	100	0.8928
OrdRec [Kor11]	gradient	100	0.8878
BMF(u,i,t) [Fre11]	MCMC	64	0.8909

Summarizing the subject of matrix factorizations, the idea of multiplying hidden variables $\mathbf{u}_i^T \mathbf{v}_j$ was chosen arbitrarily, but it turned out to be the foundation of the most accurate models in a large number of experiments carried out by many teams independently working on the problem. This gives reason to believe that the matrix factorization models are close to the optimal model, that is, to the unknown model about which we can say that it generated the collected data. Instead of using the bilinear term $\mathbf{u}_i^T \mathbf{v}_j$ we can try modelling interactions between the hidden variables nonlinearly. This opportunity is discussed in section 4.4. In experiments of many teams the most effective nonlinear methods were variants of RBM [Sal07a], described in section 4.4.1. Although the accuracy of individual RBM methods was worse than the accuracy of matrix factorization variants, RBM improved ensembles of matrix factorizations, and we can conclude, that RBM models learn some aspects of data that are not captured by matrix factorizations (but the improvement can be partially explained by the capability of modelling probability of each output $1-5$, and not necessarily by the structure of hidden variables).

4.4 Nonlinear dimensionality reduction methods

The bilinear model of regularized SVD, its clipped versions, and other types of generalized matrix factorizations contain a bilinear form of the hidden variables $c_i + d_j + \sum_{k=1}^{K} u_{ik} v_{jk}$. Combining continuous hidden variables linearly was the foundation of the most accurate models found for the Netflix task (the nonparametric experiments in section 4.2.5 also to some extent justify using the bilinear model), but it is not the only option. This section lists the most common nonlinear alternatives. Differently expressed user preferences and item features will be considered, with different choices of the output variable. Among the nonlinear methods, Restricted Boltzmann Machines gave most improvement to ensembles of methods, and this method will be described in detail in section 4.4.1. Discovering the structure of hidden variables is in general sufficiently difficult in linear models [Row99, Per10], the more difficult if we consider non-linear cases. In the Netflix Prize task it was

hard to find another way than exploring possible models by trial and error, guided by the resulting accuracy, individual and in combination with the ensemble.

The choice of the output variable for the data was discussed in section 4.2.2. The most common choice was modelling the output as a Gaussian variable, clipped or transformed by a sigmoidal function. Another choice is to model the output with a binomial distribution. Yet another choice, used e.g. in RBM, is to use a multinomial output, and model the probability of each rating 1-5 separately. The methods listed in this section use mainly the multinomial output, but Gaussian visible units were used also in RBM [Tos08b].

A natural and well performing framework for collaborative filtering is multitask learning, with per-user tasks of predicting the user ratings for all items. In such multitask models we have a set of variables expressing individual user preferences, separate for each user, and a set of variables for each item, shared by all users. One can also consider the analogous, "flipped" multitask learning setting with per-item tasks. Various types of regularized SVD can be understood as multitask learning methods. For example, the multitask learning method Bayesian PCA [Bis99a], a probabilistic model for collaborative filtering with Gaussian variables as user preferences, trained by expectation-maximization, is close to PMF [Sal08], probabilistic sparse matrix factorization with alternating MAP learning of parameters.

Turning to the non-linear methods, Restricted Boltzmann Machines (RBM) is a multitask model with binary hidden units (per-task variables which learn the user preferences), multinomial visible output units (there were also versions with Gaussian hidden units [Sal07a] or with Gaussian visible units [Tos08b, Pio09]), and a set of per-item vectors of weights, shared by all per-user tasks. Learning the item weights is not straightforward, presents computational difficulty, because there are exponentially many configurations of hidden variables. In practice, the contrastive divergence method (CD) gives a good approximation of maximum likelihood estimation of weights.

A different realization of multitask learning is, instead to condition the generated rating on configurations of multiple hidden variables (user preferences), condition the rating on only one hidden variable (user preference), that has a role of a selector. I will describe brefly two models of this kind: PLSA [Hof99b, Hof04] and URP [Mar04].

In PLSA [Hof99b, Hof04] (PLSI [Hof99c] is a similar model), called also the dyadic aspect model [Mar04], the set of ratings for all items of a given user is drawn among a number of fixed patterns. The basic version of PLSA, used to model co-occurence of words w in documents d, had a simple form: $p(d,w) = p(w|d)p(d) = p(d)\sum_z p(w|z)p(z|d)$ or equivalently $p(d,w) = \sum_z p(z)p(d|z)p(w|z)$. In [Hof99a] learning w and z by Expectation-Maximization was proposed, in a Tempered EM version (TEM). A PLSA version adapted for rating prediction [Hof03], called also the triadic aspect model [Mar04], has the form: $p(r|u,i) = \sum_z p(r|i,z)p(z|u)$, with a multinomial distribution $p(z|u)$ of the hidden selector variable z, and with a Gaussian [Hof03] or multinomial [Mar04] output rating distribution $p(r|i,z)$, for a given item i and hidden variable z. PLSA does not give satisfying accuracy on the Netflix Prize task. In my experiments, a variant of PLSA with multinomial output, learned with TEM, with annealing the learning rate and MAP step instead of ML (penalized EM – regularized estimates) gave $\text{RMSE}_{15} = 0.9449$ and did not improve the ensemble.

In the User Rating Profile (URP) model [Mar04], similarly as in PLSA, ratings are generated through choosing among a fixed number of rating patterns (distributions of all ratings), but in URP the rating pattern, called there the user attitude, is not drawn once to generate all ratings of a user, but is drawn for each item separately. The generative process in URP is following: for each user sample the vector of parameters θ from a Dirichlet prior (the prior is shared among all users), then for each item i draw z_i from the multinomial distribution $p(z_i|\theta)$ and generate the rating from a multinomial distribution

$p(r|z_i, i)$ (estimated jointly for all users for every item i). A similar model to URP, but with a uniform prior instead of the jointly learned Dirichlet prior over user attitudes, was called the vector aspect model [Mar04]. Variational Bayesian inference was used in [Mar04] for learning the URP model. URP was one of the most accurate methods among the compared in [Mar04] on the EachMovie and GroupLens datasets. I have not experimented with URP on the Netflix dataset in this work.

The Bi-LDA model [Por08] combines a user-wise LDA [Ble03, Mar04] model with a movie wise LDA. Bi-LDA assigns to each user a multinomial distribution on user-side clusters, and assigns to each movie a multinomial distribution on movie-side clusters. To generate each rating the user cluster and the movie cluster are drawn from their respective multinomial distributions, and the rating is drawn from a multinomial distribution corresponding to the pair of clusters. Bi-LDA had RMSE= 0.933 on the Netflix Prize dataset [Por08]. A related, but better performing model is Biased LITR [Har11].

In addition to probabilistic models, simplified neural-networks-like approaches are possible, which define the predicted rating as a chosen function of unknown parameters, and the parameters are learned by minimizing a regularized cost function. I implemented several neural-network models, which did not improve ensemble accuracy – I skip their description.

In the following section I describe in more detail the RBM model [Sal07a], which improved accuracy of many ensembles [Bel07c, Bel08, Tos08b, Kor09b, Tos09, Pio09], and improved also the ensemble in this work, in a directed version. Especially good accuracy was obtained by postprocessing RBM methods by variants of K-NN [Bel07c]. The remaining nonlinear methods listed above are less accurate and are not known to visibly improve the ensembles for the Netflix task, but of course for the Netflix data we cannot exclude the existence of some model with nonlinearly combined hidden variables, which, with the right learning method, could have superior accuracy to matrix factorizations. The mentioned well performing combination of RBM with K-NN is a suggestion where to look for such a model.

4.4.1 Restricted Boltzmann Machines

Using RBM for collaborative filtering tasks was proposed in [Sal07a]. In its most basic form, a Restricted Boltzmann Machine is a probabilistic model consisting of M visible binary (0-1) variables v_j and K hidden binary variables h_k. The set of MK weights w_{jk} specifies the joint probability

$$p(\mathbf{v}, \mathbf{h}|W) \propto \exp(-\sum_{j=1}^{M}\sum_{k=1}^{K} w_{jk} v_j h_k)$$

To estimate the parameters W the likelihood of the observed data vectors $\mathbf{v}$ is needed: $p(\mathbf{v}|W) = \sum_{\mathbf{h}\in H} p(\mathbf{v}, \mathbf{h}|W)$. For a given data vector $\mathbf{v}$, the likelihood $p(\mathbf{v}|W)$ could be calculated by summing over the 2^K configurations of hidden variables, but the computational complexity of this method would be too large for the K chosen in our use (up to $K = 200$).

In turn, the conditional probabilities are easy to calculate:

$$p(h_k = 1|\mathbf{v}, W) \propto \exp(-\sum_{j=1}^{M} w_{jk} v_j)$$

and

$$p(v_j = 1|\mathbf{h}, W) \propto \exp(-\sum_{j=1}^{M} w_{jk} h_k)$$

It allows to sample from the joint distribution $p(\mathbf{v}, \mathbf{h}|W)$ with an MCMC method.

Now how to learn the parameters W. Let's assume, that our observed data D is a set of vectors $\mathbf{v}^{(i)}$ for $i = 1, ..., N$, and that we want to learn the parameters W as maximum a-posteriori (MAP) point estimates, that is by maximizing with respect to W the function:

$$p(W|D) \propto p(W)p(\mathbf{v}^{(1)}, ..., \mathbf{v}^{(M)}|W) = p(W) \prod_{i=1}^{M} \sum_{\mathbf{h} \in \{0,1\}^K} p(\mathbf{v}^{(i)}, \mathbf{h}|W)$$

It was shown, that, with some assumptions, approximate estimation of the output of a similar RBM model is NP-hard [Lon10], and a similar intractability result was shown for approximate sampling.

In practice, for fixed real-world data, learning the parameters W by maximizing the approximate log-likelihood function with gradient ascent works well (it is also possible to add regularization by assuming a non-flat prior $p(W)$):

$$\frac{\partial \log p(\mathbf{v}^{(i)}|W)}{\partial w_{jk}} = \frac{\partial \log \sum_{\mathbf{h}} \exp(-\mathbf{v}^{(i)T} W \mathbf{h})}{\partial w_{jk}} - \frac{\partial \log \sum_{\mathbf{v},\mathbf{h}} \exp(-\mathbf{v}^T W \mathbf{h})}{\partial w_{jk}}$$

$$= v_j^{(i)} \; p(h_k = 1|\mathbf{v}^{(i)}) - p(v_j = 1, h_k = 1)$$

Here appears a difficulty with calculating the second term $p(v_j = 1, h_k = 1)$ in the gradient ascent. One could use Gibbs sampling, but it has large computing time. In [Hin02] the method of contrastive divergence (CD) was proposed, which is to use a very small number of sampling iterations inside one iteration of gradient ascent. Sampling is initiated with the current observation $\mathbf{v}^{(i)}$, and then even using only one iteration of Gibbs sampling appears to work well in practice.

Instead of using contrastive divergence, it is possible to use Gibbs sampling $p(\mathbf{v}, \mathbf{h}|W)$ common for all gradient ascent updates, for all subsequent observation vectors, ignoring that the weights W change during the sampling (also the pattern of missing data changes). I have not tested this method on the Netflix Prize data.

Enhanced versions of the above RBM model were used for the Netflix Prize task. In [Sal07a] the use of RBM for collaborative filtering was proposed, as a multitask model, with each task being learning all ratings of one user, and with weights W shared among all tasks (a symmetric setting with one task per movie is also possible). Each user has a separate set $\mathbf{h}_i$ of hidden variables, and the weights w_{kjl}, connecting the hidden variables h_{ik} with the outputs v_{jl}, are shared by all users. The outputs v_{jl} represent the observed rating for a movie j, and are generated from a multinomial distribution, with exactly one binary variable v_{jl} ($l \in 1..5$) set to 1. Comparing RBM with SVD, the hidden variables $\mathbf{h}_i$ correspond to user preferences, and the weights W correspond to movie features, except that there are 5 times more movie parameters in RBM than movie parameters (features) in SVD with the same K. The increased number of parameters allows to model patterns in ratings – we observed in section 4.2.2, that there are individual per-user patterns in data that cannot be generated by single-parameter or two-parameter output modelling.

The RBM model in [Sal07a] was enhanced by adding biases for visible units, adding biases for hidden units, and also enhanced by using dimensionality reduction to regularize the parameters of W (called "Factored RBM"), and making use of the missing data patterns to improve modelling of hidden variables – a method called Conditional RBM. I will outline now the concept of Conditional RBM [Sal07a].

It turns out, that a significant improvement of RMSE on the hold-out set is obtained by using the information contained in the structure of missing data. Conditional RBM [Sal07a] conditions the hidden variables h_{ik} on a binary vector $\mathbf{s}_i$ indicating which variables are observed by user i. The influence of each observed item on hidden variables is modelled

using an additional set of weights b_{jk}, incorporated into the sum $p(h_{ik}|\mathbf{v}, \mathbf{b}, W) = \sigma((\mathbf{v}_i \circ \mathbf{s}'_i)^T W_k + \mathbf{b}^T \mathbf{s}_i)$, where $\sigma(x) = 1/(1 + e^{-x})$. A similar enhancement was applied to SVD-type models (see section 4.3.5 "Improved user preferences"). The article [Kor09b] proposed adding conditional visible units to RBM, and enhancements by time effects, including frequency effects.

In rating prediction the observed vectors of user ratings are sparse, and a probabilistic way of dealing with missing data is desirable. A simplified approach would be assuming that data is missing uniformly at random and integrating out the missing data. An approximation proposed in [Sal07a] is simply ignoring the weights leading to the missing items while sampling the hidden variables (this method works well in practice, but, as noted in [Hin10], the result is not precisely correct). As outlined in section 3.4, the problem of missing data is more complex. [Mar08] proposed a model called cRBM/E-v, which to some extent corrects predictions of the Conditional RBM model by using additional biases, active on missing values.

To speed up learning with contrastive divergence I developed an RBM version with directed weights. The idea is that weights learned by contrastive divergence, even with only one step of Gibbs sampling, give hidden variables good enough to make accurate predictions, provided that a separate set of weights is used to predict the outputs from the hidden variables. In the directed RBM two separate sets of weights are learned: W^{up} weights from visible nodes to hidden nodes and W^{down} from hidden nodes to visible. The conditional probabilities are following:

$$p(h_k = 1|\mathbf{v}, W^{up}) \propto \exp(-\sum_{j=1}^{M} w_{jk}^{up} v_j)$$

$$p(v_j = 1|\mathbf{h}, W^{down}) \propto \exp(-\sum_{j=1}^{M} w_{jk}^{down} h_k)$$

In the proposed directed RBM: let $\mathbf{v}_0 = \mathbf{v}^{(i)}$ (with the missing values in $\mathbf{v}^{(i)}$ set to zeros), $\mathbf{h}_0$ is sampled from $p(\mathbf{h}|\mathbf{v}_0, W)$, $\mathbf{v}_1$ is sampled from $p(\mathbf{v}|\mathbf{h}_0, W)$, and $\mathbf{h}_1$ is sampled from $p(\mathbf{h}|\mathbf{v}_1, W)$. Then the updates of W^{up} with the CD method have the form:

$$W^{up} \mathrel{+}= \eta(\mathbf{v}_0 \mathbf{h}_0^T - \mathbf{v}_1 \mathbf{h}_1^T)$$

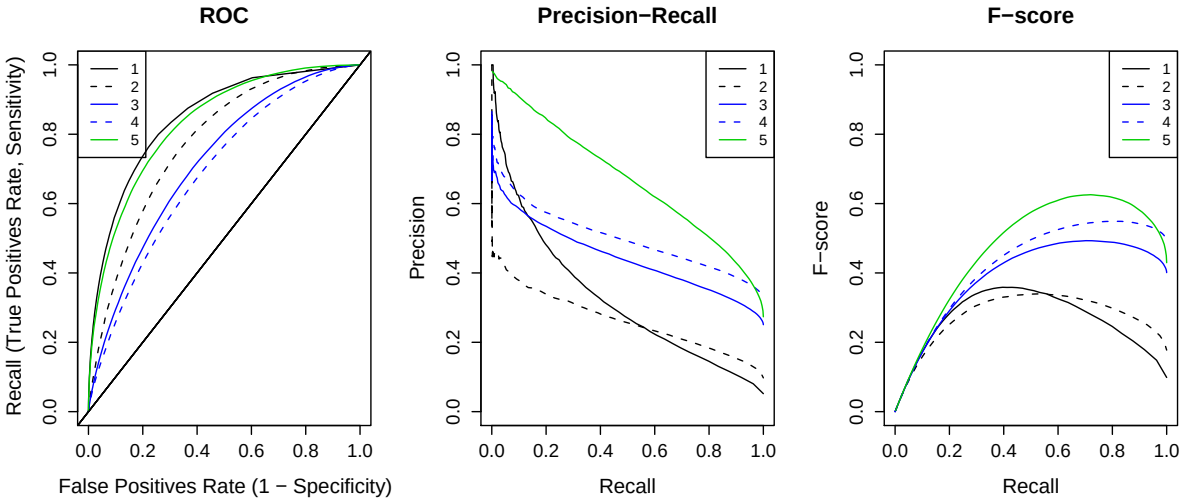
The W_{down} weights are learned to reconstruct the observed data with highest probability, with $\mathbf{h}^0$ treated in the moment of learning as fixed. Maximizing the likelihood of the observed data gives a rule similar to wake-sleep [Hin95]:

$$W^{down} \mathrel{+}= \eta(\mathbf{v}_0 - \mathbf{v}_1)\mathbf{h}_0^T$$

The directed RBM methods improved RMSE by about 0.0002-0.0003 in comparison with an analogous undirected RBM learned by contrastive divergence with $T = 1$. Three versions of the directed RBM were part of the final ensemble, listed also on the bottom of the table 29. DRBM was a conditional version. DRBM2 and DRBM3 were unconditional versions with different sets of parameters learning rate and weight decay. Comparing with the RBM from [Sal07a], DRBM was not factored, used weight decay, used decreasing learning rate, similarly to simulated annealing methods, and contained biases for visible units (five for each movie), but did not contain biases for hidden units. DRBMKNN in table 29 is DRBM postprocessed by my modification of KNNMovieV3 [Tos08b], described in section 4.5.1.

Because RBM methods give as a result probability distribution of ratings, we can examine classification performance of prediction of each rating $1 - 5$. Figure 33 shows

Figure 33: DRBM classification accuracy.



three plots of the results of DRBM: receiver operating characteristic (ROC), precision-recall plot, and F-score vs. recall.

Good results were obtained by postprocessing RBM with K-NN methods. The best single method in the ensemble [Bel07c] was RBM postprocessed by a K-NN variant with jointly derived weights [Bel07c].

Hidden variables were binary and the observed variables were multinomial in [Sal07a, Mni10], but using other distributions is possible, e.g. Gaussian observed variables or Gaussian hidden variables. CRBM with Gaussian observed variables were used in [Bel07c, Tos08b].

Table 29 lists RBM implementations by various authors.

Table 29: RBM variants.

Method	Learning	K	RMSE_{15}	RMSE_{quiz}
Factored CRBM [Sal07a]	CD	500,30		∼0.9050
CRBM [Bel07c]	CD	200		0.9029
CRBM + KNN [Bel07c]	CD	100+50		0.8888
CRBM Gaussian [Bel07c]	CD	256		0.9056
CRBM + time-dep. KNN [Bel08]	CD	100+35		0.8893
CRBM [Tos08b]	CD	150		0.9057
CRBM Gaussian [Tos08b]	CD	200		0.9056
CRBM + KNN [Tos08b]	CD	100+55		0.8900
CRBM + KNN + KNNMovieV3 [Tos08b]	CD	150+55+122		0.8832
CRBM [Pio09]	CD	200		0.9020
CRBM w/time (trbm100) [Pio09]	CD	100		0.9036
CRBM w/frequency (frbm200) [Pio09]	CD	200		0.9002
CRBM w/time and frequency [Kor09b]	CD	200		0.8928
DRBM	CD + gradient	100	0.9101	(∼0.9130)
DRBM2	CD + gradient	100	0.9175	(∼0.9205)
DRBM3	CD + gradient	100	0.9192	(∼0.9220)
DRBM+KNN	CD + gradient	100+50	0.8888	(∼0.8920)

The experience of many teams working on the Netflix Prize task was that including RBM methods significantly improves accuracy of ensembles [Bel07c, Kor08, Tos08b, Kor09b, Tos09, Pio09], which shows that RBM capture some aspects of data uncaptured

by other methods.

There are many questions left about RBM methods. Is the underlying computational complexity necessary? Does any similar model to RBM exist that has close accuracy, but with smaller computational complexity? Do RBM have equivalent forms with matrix regularization, as regularized SVD variants are equivalent to matrix trace norm regularizations? What is the kernel (dual) version of the RBM model or similar models?

An untested idea is to model jointly the ratings $p(\mathbf{r}_i|\mathbf{h}_i, W)$ and the structure of missing data $p(\mathbf{s}_i|\mathbf{h}_i, W)$, where $\mathbf{r}_i$ is the vector of user ratings (further split into observed ratings, and missing ratings), $\mathbf{s}_i$ is the vector of binary indicators of which movies were rated, $\mathbf{h}_i$ is a vector of hidden user preferences, and W is the matrix of movie features, shared by all users. The predicted probability of selecting the item to rate could help in ranking missing items to create lists of personalized recommendations. Because predictions are made for unobserved data, it would be helpful to tune the model using a sample of rated random items, if such data were available.

4.5 Distance-based methods

After discussing regularized SVD and nonlinear dimensionality reduction methods, now I turn to describing a large group of algorithms, distance-based methods, which can also be called similarity-based methods (the notion of distance will be used here informally, not necessarily satisfying the metric axioms). The idea behind is that some observations are distant from each other, and will take unrelated values or negatively correlated values, and some observations are close, and will take similar values. The reviewed distance-based methods are divided into two groups: K-nearest neighbor methods and kernel based methods. The predicted values are ratings, residuals of global effects or residuals of more complex methods (postprocessing by distance-based methods was able to improve most of other families of algorithms).

A similar need to augment methods by using local similarities can be observed in other tasks than collaborative filtering. Even in simple regression models, besides using regression coefficients, biases, and dimensionality reduction (see, for example, Factorization Machines [Fre11]), it is sometimes convenient to use similarities between observations (for example, errors correlated in time) and similarities between features (for example, effects for groups of similar features).

The work [Bel07a] "Modelling relationships at multiple scales" distinguished three levels of modelling effects in the Netflix data:

- global effects, present in the entire data or in large parts of data, for example, biases,
- middle-level effects, captured by methods performing dimensionality reduction, for example, SVD,
- local effects, capturing large relationships among similar movies – distance-based methods belong to this layer, with a few exceptions.

Among the most accurate methods for the Netflix Prize dataset the following pattern can be noticed: the global effects layer has $O(N+M)$ parameters, the dimensionality reduction layer has $O(KNM)$ parameters, and the neighborhood layer has $O(M^2)$ parameters, where N is the number of users, M is the number of movies, and K is the dimension in dimensionality reduction (all bounds with a remark, that modelling the time effects can increase the number of parameters up to several tens of times). Both K-NN and kernel methods are good at capturing (explaining) local similarities, that is correlations for very similar movies, as opposed to middle-level effects, where the dimensionality reduction used does not allow to capture a large number of different local relationships, even when those relationships are very significant.

Often in various applications it is easy to construe content-based similarity between items. Such similarity information is often local – for each item we can recognize only a few similar items. In such cases, if we do not have additional data about users' preferences, distance-based methods are likely to be more accurate than methods performing dimensionality reduction.

In the following discussion I will focus mostly on distances (or similarities) between movies. Some authors considered also distance between users, but those methods contributed less to the accuracy of ensembles, and they need more computation time. It would be best to use the distance between observation pairs (user, movie), but such methods would have too large computational complexity.

4.5.1 K-nearest neighbors

K-NN methods are commonly chosen as a collaborative filtering algorithm inside recommender systems. Their advantage is explainability, intuitive explanation of predictions, such as: "the movie W is recommended to you, because you liked the movies X,Y,Z". Another advantage is that K-NN methods do not require precise identification of the underlying model of the data. K-NN methods often work satisfactorily well "out of the box", or using relatively small amount of parameter tuning, which saves developer's time. On rating data the usual drawbacks of K-NN in comparison with SVD-based methods are: worse predictive accuracy, larger computational complexity and larger memory usage.

When applied to collaborative filtering, K-NN methods can be divided into two basic types: using item-item or user-user similarities. I will focus here on the item-item approach, which was more accurate than the user-user approach on the Netflix Prize task (measuring accuracy as individual methods and also in connection with the ensemble). An additional advantage is that item-item methods are much faster than user-user, because in the Netflix data we have about 40 times more users than items. A third type of K-NN method can be also considered, using similarity between observations (user-item pairs). One such method will be described at the end of this section.

The most common K-NN form used for collaborative filtering is to choose a similarity measure between items (or users, but we focus here on the item-item view), and use the similarity to select the nearest items, and to weight the items. Predictions are made by a weighted average of residuals [Her00a]:

$$\hat{r}_{ij} = \frac{\sum s_{j_2 j} r_{ij_2}}{\sum_{j_2 \in N(i,j)} s_{j_2 j}} \tag{37}$$

where $s_{j_2 j}$ is the similarity between movie j_2 a movie j, and $N(i,j)$ is the set of K movies with the largest similarity to movie j among the movies rated by user i. Most commonly chosen as the similarity measure were variants of Pearson correlation between common observations [Sar01] (called also cosine similarity), on the centered data, that is with the movie mean removed $y_{ij} = r_{ij} - \mu - d_j$, or generally, with global effects removed:

$$s_{j_2 j} = \frac{\sum_{i \in I_{j_2 j}} y_{ij_2} y_{ij}}{(\sum_{i \in I_{j_2 j}} y_{ij_2})(\sum_{i \in I_{j_2 j}} y_{ij})}$$

A big improvement of accuracy of K-NN with Pearson similarity is obtained by regularization (shrinking) of similarities [Bel07a, Tos08b]:

$$\tilde{s}_{j_2 j} = s_{j_2 j} \frac{|I_{j_2 j}|}{|I_{j_2 j}| + \lambda}$$

A similar effect to regularization was obtained by using a lower bound estimator of Fisher Z-transform of unregularized Pearson correlation [NF07, Tak07a]. We see here that se-

lecting nearest neighbors is a ranking problem, and involves both using regularization (selecting the right prior), and correcting for the confidence of the calculated similarities.

Another improvement of accuracy was obtained by clipping correlations to non-negative values [Tak07a, Bel07a, Tos08a].

Among the non-model-based techniques (like the described above) in my experiments best worked the KNNMovie version [Tos08a, Tos08b], which applies a logistic transformation to regularized Pearson or set correlations, clipped to non-negative values. In the version KNNMovieV3 [Tos08b], including the distance in time between observations, similarities inserted into the formula (37) have the form:

$$\overline{s}_{j_2 j} = \sigma(\alpha + \beta \tilde{s}_{j_2 j} \exp(-\gamma |t_j - t_{j_2}|))$$

where $\sigma(x) = 1/(1 + e^{-x})$.

In my experiments, better accuracy improvement gave penalizing the distance in time with a term taken from the KRRT method (KRRT is described in section 4.5.3):

$$\overline{s}_{j_2 j} = \sigma(\alpha + \beta \tilde{s}_{j_2 j} - \gamma \log(1 + |t_j - t_{j_2}|))$$

The parameters $\alpha, \beta, \gamma, \lambda$ in my implementation of K-NN were automatically optimized with the Praxis procedure from the Netlib library, to minimize RMSE_{15}. Postprocessing SVDMT2 with 30 features (see section 4.8.1) by the above K-NN version gave the best accuracy among all methods implemented by me: $\text{RMSE}_{15} = 0.8819$.

Instead of calculating Pearson item-item correlation on centered ratings good results gave using Pearson correlation (cosine similarity) between reduced dimensionality representations of movies in the form of normalized $\mathbf{v}_j$ vectors from SVD [NF07, Pry98] (likewise we can calculate SVD-based user-user similarities $\mathbf{u}^T\mathbf{u}$ [Bel07b]). In [Pat07] one nearest neighbor was used, and this method was included in the final ensemble in this work. The ensemble also included prediction by the arithmetic average of normalized ratings of 20 nearest neighbors, using a version of Pearson correlation.

Apart from Pearson correlation, a popular choice of similarity measure was set correlation [Tos08b], that is the normalized count of common users:

$$s_{j_2 j} = \frac{|I_{j_2 j}|}{|I_{j_2}||I_j|}$$

and its regularized version:

$$\tilde{s}_{j_2 j} = \frac{|I_{j_2 j}|^2}{|I_{j_2}||I_j|(|I_{j_2 j}| + \lambda)}$$

Good performance of the set similarity is an additional manifestation of the large amount of information carried in the missing data structure.

There were also other propositions for the similarity measure, such as the MSE similarity [Pio09, Tos09], that is the sum of squared differences between normalized ratings.

The above approaches have the disadvantage, that the similarity function is chosen arbitrarily. An approach proposed in [Bel07a], which gives better predictive accuracy, exploits the multi-task structure of the collaborative filtering task, where each user is treated as a separate task. The multi-task structure allows for joint optimization of weights. In [Bel07b] this idea was realized by separate per-prediction regressions, which calculated the best similarity weights by a least squares fit, with regularization, and with weights constrained to be non-negative. The prediction rule in [Bel07b] is:

$$\hat{r}_{ij} = \sum_{j_2 \in N(i,j)} s_{j_2 j} r_{i j_2}$$

where weights $s_{j_2 j}$ are calculated by non-negative least squares minimization:

$$min_{s_{j_2 j}>=0} \sum_{i \in I_j} (r_{ij} - \hat{r}_{ij})^2$$

with additional regularization. Each rating is predicted using a separate ridge-regression-like calculation – the per-prediction weights $\mathbf{s}$ are calculated by solving a system of linear equations:

$$A\mathbf{s} = \mathbf{b}$$

where

$$I_{j_1 j_2} = \frac{\sum_{U(j_1,j_2)} r_{ij_1} r_{ij_2} + \beta * avg}{|U(j_1, j_2)| + \beta}$$

$$\mathbf{b}_{j_1} = \frac{\sum_{U(j,j_1)} r_{ij} r_{ij_1} + \beta * avg}{|U(j, j_1)| + \beta}$$

where $U(j_1, j_2)$ is the set of users who rated both movies j_1 and j_2. Additionally, the weights $\mathbf{s}$ are subjects to non-negativity constraints $s_{j_2 j} \geq 0 \ \forall_{j_2,j}$.

The K-NN family is related to other collaborative filtering methods. Multitask K-NN with jointly learned weights, such as the above Bellkor's K-NN version, are close to per-movie linear models (see section 4.5.2) and to regression on similarity [Tos08b]. In turn, factorized per-movie linear models [Kor08] and regression on factorized similarity [Tos08b] are related to NSVD and SVD++ methods (see section 4.3.5). K-NN are also generally related to kernel methods (section 4.5.3), as the similarity function in K-NN corresponds to covariance (kernel) in multitask Gaussian Processes approaches. In kernel methods it is essential to choose well the monotone function transforming the product $x_j^T x_{j_2}$ into a kernel $f(x_j^T x_{j_2})$. In turn, in K-NN methods the set of nearest neighbours does not change with different choices of the transformation f of the similarity, hence K-NN methods seem to be more resistant to inaccuracies in defining similarity. When we do not know much about the model that generated the data (and this is typically the case when starting the analysis of real-life data), K-NN may be a good choice to obtain good enough accuracy without putting much effort into developing a prediction method. The advantages of K-NN are smaller computational complexity and easier parameter tuning.

A disadvantage of K-NN is that methods based on the right probabilistic model have better attainable accuracy. In my experience (and also others - [NF07]), adding to the ensemble Kernel Ridge Regression with a Gaussian kernel caused that most K-NN methods did not give a significant contribution to the ensemble anymore (K-NN methods improved accuracy when KRR was not present in the ensemble).

Table 30 comapres variants of K-NN implemented by various authors. On the Netflix data, in experiments of multiple teams, user-user K-NN gave worse accuracy than item-item approaches.

K-NN algorithms use for prediction only the most correlated movies, ignoring the user ratings for remaining movies, and those skipped ratings carry useful information. Joe Sill mentioned in a blog article a K-furthest neighbors algorithm, where predictions were based only on the most negatively correlated movies, obtaining $\text{RMSE}_{quiz} = 0.9562$.

K-NN methods work well in connection with dimensionality reduction methods, which are unable to model a large number of local effects. Several ways to combine K-NN with other methods were proposed, from applying K-NN on residuals of a given method (the most common way) to construing integrated models. K-NN on residuals of RBM was the best method in the ensemble [Bel07c] with $\text{RMSE}_{quiz} = 0.8888$.

Another way to combine the methods [Tak08c] was to simultaneously learn a matrix factorization model and K-NN, with one method weighted by a constant λ and the other weighted by $1 - \lambda$.

Table 30: Standalone KNN methods.

Method	K (NN)	RMSE_{quiz}
Pearson-KNN on global effects [Bel07b, Bel07d]	20	0.9364
KNN, jointly derived weights [Bel07b, Bel07d]	50	0.9174
KNN, jointly derived weights, item-item [Bel07a]	50	0.9075
KNN, jointly derived weights, user-user [Bel07a]	50	0.9180
KNN, jointly derived weights [Bel07b]	50	0.9082
KNN, jointly derived weights, w/o global effects [Bel07c]	25	0.9496
Bin-KNN/CorrSim [Bel07c]	50	0.9215
Corr-KNN [Bel07c]		0.9170
Mse-KNN [Bel07c]		0.9237
Supp-KNN [Bel07c]		0.9335
Pearson-KNN [Tos08b]	30	0.9229
KNNMovieV3 [Tos08b]	55	0.9102
KNNMovieV4, jointly derived weights [Tos08b, Bel07b]	50	0.9112
NB [Tak07a]	16	0.9313
NB-CORR [Tak08b]	15	0.9280

To get the most accurate prediction we have to properly fully integrate near neighbor models with the matrix factorization into one model. Such methods were listed in section 4.8.2 "Integrated models". One way to do it is augmenting SVD with local item-item similarity information [Bel07a, Bel07c]. Another way is to add a neigborhood modelling component to the SVD formula. Here best effects gave merging the matrix factorization with two linear models (they can be seen as K-NN with shared $O(M^2)$ weights) [Kor08].

Good results were obtained [Pio09, Tos09] by tuning parameters of a method to optimize RMSE of the whole ensemble, instead of optimizing individual RMSE's. [Pio09] emphasised that tuning neighborhood models this way gave especially large improvement of the ensemble accuracy.

Table 31 lists chosen K-NN methods applied to residuals of other methods, K-NN methods that use similarity based on SVD features, and factorized K-NN methods. The K-NN methods on residuals of RBM were moved to table 29.

At the bottom of the table are listed the methods from my ensemble. KNN0 is 1-NN prediction, using as distance the cosine similarity between SVD features. KNN1 is KNN0 weighted by exponential of the SVD-based distance. KNN20 is average residual of 20-NN with distance as in KNN1. SVDMT2KNN is SVDMT2 (see section 4.3.6) postprocessed with the modified KNNMovieV3 [Tos08b], described above.

On the Netflix Prize task mainly item-item K-NN was used, and occasionally user-user K-NN. One other opportunity was left unexplored – similarity between observations (user-item pairs). I will describe a heuristic approach, which can be classified as a near-neighbour method based on similarity between observations, and which is fast enough to run it on the Netflix data (although the experiment was performed only on a small subset of the Netflix data). In regularized SVD the prediction for a given feature k, if assuming a constant regularization λ, and estimating one variable at a time, has the form:

$$u_{ik} = \frac{\sum_{j_2} v_{j_2 k} y^{(k)}_{ij_2}}{\sum_{j_2} v^2_{j_2 k} + \lambda} \qquad v_{jk} = \frac{\sum_{i_2} u_{i_2 k} y^{(k)}_{i_2 j}}{\sum_{i_2} u^2_{i_2 k} + \lambda}$$

Now the heuristic idea is to assume, that $u_{ik}v_{jk}$ explains a fixed percent C of the residual y_{ij}: $u_{ik}v_{jk} \approx C y^{(k)}_{ij}$. (the index k will be skipped, and y_{ij} denotes a residual in the current iteration). Roughly approximating:

$$u_i v_j \approx \frac{\sum_{i_2 j_2} y_{ij_2} y_{i_2 j} u_{i_2} v_{j_2}}{\sum_{i_2 j_2} (u_{i_2} v_{j_2})^2 + \lambda_2} \approx C_2 \frac{\sum_{i_2 j_2} y_{ij_2} y_{i_2 j} y_{i_2 j_2}}{\sum_{i_2 j_2} y^2_{i_2 j_2} + \lambda_3}$$

Table 31: KNN methods as postprocessing.

Method	N features	K (NN)	RMSE_{15}	RMSE_{quiz}
MF with AUF [Bel07a]	40	50		0.8990
NNMF with AUF/MseSim [Bel07c]	128			0.8955
KNN on res. of NNMF [Bel07c]	180	30		0.8953
MF with NB correction (BRISMF) [Tak08a]	1000	∞		0.8904
NSVD1 with NB correction S1 [Tak08c]	80	∞		0.9037
HYBRID1 with NB correction S1 [Tak08c]	400	40		0.8845
SVD++ integrated with NB [Kor08]	200	∞		0.8868
RMF w/NB [Tos08a]	100	50		$\text{RMSE}_{probe} = 0.9069$
Neighborhood-aware MF [Tos09, Tos08a]	100	50		0.8856
Factorized NB [Kor10]	200	∞		0.8953
KNN0 [Pat07]	96	1	0.9525*	($\sim$0.9555)
KNN1	96	1	0.9482*	($\sim$0.9510)
KNN20	96	20	0.9389*	($\sim$0.9420)
SVDMT2KNN	30	50	0.8819	($\sim$0.8850)

* - RMSE_{15} obtained by regression with basic predictors (global effects)

After writing down the above formula using matrices, we get a fast algorithm, which turns out to have good accuracy. Each iteration k of the algorithm has the form:

$$Y_k = R - \sum_{k'=1}^{k-1} \beta_{k'} \hat{Y}_{k'} \qquad \hat{Y}_k = \frac{Y_k^T Y_k Y_k^T}{A^T (Y_k \circ Y_k)\ A^T}$$

where $\circ$ is the entrywise (Haddamard) product, and A is a 0-1 matrix indicating which ratings are observed in the training set. The coefficients $\beta_{k'}$ are repeatedly calculated using linear regression on the test set.

The algorithm can be understood as calculating a weighted average of ratings (residuals), which are "close" to the predicted rating r_{ij}, where the "close" set of observations is the data matrix clipped to movies rated by user i, and to users who rated movie j.

The algorithm was run for $K = 6$ iterations. The experiment was conducted on a small subset (see section 3.3) of the Netflix data, where the training set is exactly 50% dense.

Table 32: Near neighbor methods on the small dataset.

Method	K	RMSE_{small}
NNObs	6	0.7482

The result is compared with RMSE_{small} of other methods in chapter 5 (the best result, $\text{RMSE}_{small} = 0.7415$, was obtained by KRR).

4.5.2 Per-item linear model

A family of simple linear models turned out to improve accuracy of ensembles, and also created very accurate integrated models with matrix factorizations (see section 4.8.2).

In the work [Pat07] I used the following linear model:

$$\hat{r}_{ij} = d_j + |J_i|^{-0.5} \sum_{j_2 \in J_i} w_{j_2 j} \tag{38}$$

For the sparse Netflix Prize dataset the fastest method of learning the parameters of this model was stochastic gradient descent, here run separately for each movie, for $16 - 21$

iterations. In [Pat07] I described only the most contributing linear model, but the ensemble in [Pat07] contained several similar linear models with different constants (1, $\log|J_i|$, $|J_i|^{-0.5}$), some using residuals of ratings instead of indicators of observed ratings, some methods weighted users with the term $|J_i|^{-0.5}$ while learning, and some methods used the qualifying set for transductive learning. They are listed in table 33 as RRBA, RRBQ, RRBAS, RRBAS2, RRBAL, RRRAS, and also were included in this work's ensemble (chapter 5). The above linear model is close to the Slope One prediction [Lem05], which averages, for all movies rated by the user, over the average difference in ratings for users who rated both movies.

The work [Kor08] describes a rating-based version:

$$\hat{r}_{ij} = d_j + |J_i|^{-0.5} \sum_{j_2 \in J_i} w_{j_2 j}(r_{ij_2} - d_{j_2}) \tag{39}$$

The above linear models contribute to ensembles, but do not have good individual accuracy.

As noticed in [Kor08], well performs combining the models (38) and (39):

$$\hat{r}_{ij} = d_j + |J_i|^{-0.5} \sum_{j_2 \in J_i} w_{j_2 j} + |J'_i|^{-0.5} \sum_{j_2 \in J'_i} w'_{ij_2}(r_{ij_2} - d_{j_2}) \tag{40}$$

An implementation of this method was included in my final ensemble, called RR2.

Further improvement of (40) was obtained by taking into account the distance in time [Kor09b, Bel08]:

$$\hat{r}_{ij} = d_j + |J_i|^{-0.5} \sum_{j_2 \in J_i} w_{j_2 j} \exp(-\beta_i |t_{ij} - t_{ij_2}|) + |J'_i|^{-0.5} \sum_{j_2 \in J'_i} w'_{j_2 j}(r_{ij_2} - d_{j_2}) \exp(-\beta_i |t_{ij} - t_{ij_2}|)$$

This method is listed as "Time-aware NB" in table 33, and its modification as PQ3 [Kor09b].

Table 33 lists RMSE's of linear models given by various authors. At the bottom are listed my implemenations. Some of the results are blended with a set of six predictors BASIC2 (five regularized empirical user probabilities, and a regularized movie mean learned on the residuals of user mean; slightly different from the set BASIC used in the previous ensemble [Pat07]).

The per-item linear models have a similar form to K-NN with jointly learned weights from multiple tasks [Bel07a], except that there is no limit on the number of nearest neighbors, instead there is a normalizing factor, usually in the form $|J_i|^{-0.5}$, and also stochastic gradient descent is used for learning instead of calculating the optimal weights in one step, which is feasible for a small number of near neighbors. Because of the connection with K-NN methods, per-item linear models were called global neighborhood models in [Kor08], and I followed this convention here, placing their description in the section "Distance-based methods". The work [Kor08] notices also, that factorized versions of linear models are closely related to NSVD, and [Kor08] proposes methods with factorized similarity, binary input based and rating based, in item-oriented and user-oriented versions. The factorized methods were described in section 4.3.5 "Improving user preferences".

The per-item linear models, serving as methods that capture local item-item correlations, complemented well the matrix factorizations models [Bel08] (see sections 4.8.1 and 4.8.2).

4.5.3 Kernel methods

Multitask kernel methods model each user's vector of ratings by specifying all item-item relations through an $M \times M$ kernel matrix K. Because ratings from the Netflix dataset are

Table 33: Per-item linear models.

Method	Version	Weighting	Dataset	RMSE_{15}	RMSE_{quiz}
NB-LS-BIN [Tak08b]					0.9605
Neighborhood model [Kor08]					0.9002
Regression on sim. [Tos08b]					0.9229
Regr. on sim., w/unknown [Tos08b]					$\text{RMSE}_{probe} = 9278$
Regr. on fact. sim., item [Tos08b]					$\text{RMSE}_{probe} = 9313$
Regr. on fact. sim., user [Tos08b]					$\text{RMSE}_{probe} = 9371$
Time-aware NB [Kor09b]					0.8885
PQ3 [Kor09b]					0.8870
RRBA	binary	1	TrPrQu	1.0129	(∼1.0160)
RRBQ	binary	1	PrQu	1.0423	(∼1.0450)
RRBAS+BASIC2	binary, weight.	$\lvert J_i+1\rvert^{-0.5}$		0.9511	(∼0.9540)
RRBAS2+BASIC (LM [Pat07])	binary	$\lvert J_i+1\rvert^{-0.5}$		0.9506	(∼0.9535)
RRBAS2+BASIC2	binary	$\lvert J_i+1\rvert^{-0.5}$		0.9539	(∼0.9535)
RRBAL+BASIC2	binary	$1/log(\lvert J_i\rvert+1)$		0.9505	(∼0.9560)
RRRAS+BASIC2	ratings, weight.	$\lvert J_i+1\rvert^{-0.5}$	Tr	0.9361	(∼0.9390)
RR2 [Kor08]	binary+ratings	$\lvert J_i+1\rvert^{-0.5}$		0.9115	(∼0.9145)

well approximated by Gaussian distributions, commonly used was a multitask Gaussian Processes (GP) approach, where vectors of user ratings are assumed to be generated from $N(\mathbf{m}, K)$, where the mean $\mathbf{m}$ and the covariance K (kernel) are shared by all users. MAP estimate of the posterior distribution of GP is called Kernel Ridge Regression, and is equivalent to ridge regression for some choice of features. When there are more features than the observations, KRR is faster than the corresponding ridge regression method (it can work even in cases with infinitely many features). An advantage of the dual view of regression is also easier modelling of the similarities between items, for example changing the covariance depending on the distance in time between two observations (two ratings of a user) is easier than an analogous modification of matrix factorization.

In [Pat07] I used a per-user Kernel Ridge Regression, where the joint kernel is estimated by the cosine similarity between vector of features taken from a learned regularized SVD. The KRR variants SVDKRRG, SVDKRRG2, KRRT, KRRT2 listed in table 34 used the following variant of regularized SVD with $K = 85$ features:

$$\hat{r}_{ij} = \mu + c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j$$

When the movie features v_{jk} are learned and fixed, the user preferences u_i can be estimated by ridge regression:

$$\mathbf{u}_i = (V_i^T V_i + \lambda_i)^{-1} V_i^T \mathbf{y}_i$$

where the predicted outputs y_i are ratings with global effects removed:

$$y_{ij} = r_{ij} - \mu - c_i - d_j$$

and V_i is a slice of the vector of movie features V with rows selected by the vector of indicators of observations J_i. An equivalent dual method is prediction by kernel ridge regression (KRR):

$$\hat{y}_{ij} = \mathbf{v}_j V_i^T (V_i V_i^T + \lambda_i)^{-1} \mathbf{y}_i$$

Other kernels than the linear one $k_{j_2 j} = v_j^T v_{j_2}$ can be used in the KRR prediction equation:

$$\hat{y}_{ij} = K_{j:J_i} (K_{J_i:J_i} + \lambda_i)^{-1} \mathbf{y}_i$$

A kernel K corresponds to ridge regression with some choice of features $\phi(v_j)$, which are not defined directly. Among several tried out methods a Gaussian kernel defined on

normalized vectors from regularized SVD: $x_j = v_j/||v_j||_2$ performed best in experiments [Pat07].

$$k_{j_2 j} = \exp(C_0||x_{j_2} - x_j||_2^2) = C_1 \exp(C_2 x_{j_2}^T x_j)$$

This method is listed in table 34 as SVDKRRG with the constants $C_1 = 2.5$, $C_2 = 1$, $\lambda = 2$, and SVDKRRG2 (named SVD_KRR in [Pat07]) with the constants $C_1 = 2$, $C_2 = 2$, $\lambda = 0.5$. The number of user observations (used user ratings) was limited to 500 per user (selected were the most frequent items). Multitask kernel learning with a Gaussian kernel (called also the RBF kernel) performed also well in other applications in [Law03, Law05].

It turned out that it is easy to augment KRR with time information, largely improving both the accuracy of the method and the ensemble accuracy. Best results were obtained with the following kernel:

$$k_{j_2 j} = C_1 \exp(C_2 x_{j_2}^T x_j) - C_3 \log(1 + |t_{j_2} - t_j|)$$

This method is listed in table 34 as KRRT, with the constants $C_1 = 1$, $C_2 = 2$, $C_3 = 0.02$, $\lambda = 0.5$. After adding KRRT to the ensemble of 56 methods from [Pat07] RMSE of the ensemble improved by about $0.6\% - 0.7\%$ (measuring improvement comparing to Cinematch). The improvement was large mainly because the methods in [Pat07] did not use the time information.

The KRRT2 method in table 34 is, in turn, a linear kernel with the following time modifier:

$$k_{j_2 j} = x_{j_2}^T x_j * (1 - C_3 \log(1 + |t_{j_2} - t_j|))$$

where t_j, t_{j_2} are the dates of ratings j, j_2. In KRRT2 the following constants were chosen: $C_3 = 0.05$ and $\lambda = 2$.

Another possibility is using the structure of missing data. I used here the results of an algorithm similar to SVD++ [Bel07e]. The biases and the NSVD part from the SVD++ model served as global effects for KRR, giving the following residuals to be predicted by KRR:

$$y_{ij} = r_{ij} - (\mu + c_i + d_j + |J_i|^{-0.5}(\sum_{j_2 \in J_i} \mathbf{w}_{j_2})^T \mathbf{v}_j) \tag{41}$$

On the above residuals the KRR variants were learned listed in the table 34 as KRRS2 and KRR3.

KRRS2 was KRRT based on a regularized SVD++ with $K = 100$ features. The biases were learned as a preprocessing, the U and V variables were learned by alternating least squares, all features at a time, and the weights W in the NSVD1 part were learned by gradient descent. The number of ratings per user in KRRS2 was limited to 700.

KRR3 combined four different kernels and two ways of correcting the kernel for the dates of ratings:

$$k_{j_2 j} = (C_1 * v_{j_2}^T v_j + C_2 * v_{j_2}^{'T} v'_j + C_3 * \exp(2 * (x_{j_2}^T x_j - 1)) +$$

$$+ C_4 * \exp(2(x_{j_2}^{'T} x'_j - 1)) * (1 + C_5[t_{j_2} = t_j][j_2 \neq j]) - C_6 \log(1 + |t_{j_2} - t_j|)$$

where $\mathbf{v}_j$ come from a variant of SVD++ ($\text{RMSE}_{15} = 0.8947$; description skipped) with features learned by alternating least squares, and with a special version of the NSVD term. $\mathbf{v}'_j$ are the 100 first eigenvectors of the regularized empirical covariance matrix: $Cov_{j_2 j} = (\sum_{i \in I_{j_2 j}} y_{ij} y_{ij_2})(|I_{j_2 j}| + 1000)$, with the negative values set to zero, and with the diagonal set to zero, and the eigenvectors are multiplied by the square roots of the corresponding eigenvalues. The y_{ij} values, on which the covariance matrix is calculated, are residuals of biases and the NSVD term. x_j and x'_j are normalized features $\mathbf{x}_j = \mathbf{v}_j/||\mathbf{v}_j||_2$, and $\mathbf{x}'_j = \mathbf{v}'_j/||\mathbf{v}'_j||_2$. The manually picked constants were $C_1 = C_2 = C_3 = C_4 = 0.45, C_5 = 1.0, C_6 = 0.08, \lambda = 0.5$.

The SVDKCOV method is similar to KRR3, but combined two kernels:

$$k_{j_2 j} = (C_1 * v'^T_{j_2} v'_j + C_2 * \exp(C_3 * (x^T_{j_2} x_j - C_4))) * (1 + C_5 [t_{j_2} = t_j][j_2 \neq j]) - C_6 \log(1 + |t_{j_2} - t_j|)$$

where $\mathbf{x}_j = \mathbf{v}_j / ||\mathbf{v}_j||_2$, where v_j are 30 features from the method SVDMT, described in section 4.3.3, and v'_j are 70 features coming from the eigen decomposition of the covariance matrix of residuals of SVDMT. The constants in SVDKCOV were chosen by automatic parameter tuning by the Praxis procedure, giving, after rounding: $C_1 = 0.9, C_2 = 0.03, C_3 = 6.8, C_4 = 0.45, C_5 = 0.75, C_6 = 0.01$. In the KRR in [Pat07] constant regularization was used, but further experiments showed that a little better worked linear regularization: $\lambda = \lambda_1 + \lambda_2 |J_i|$. For SVDKCOV automatic parameter tuning selected the following regularization parameters (after rounding): $\lambda_1 = 16$ and $\lambda_2 = 0.032$.

Instead of using features from SVD, the parameters X in the reduced dimensionality representation of the kernel can be learned in the multitask setting by gradient descent [Law09]. For the Gaussian kernel $k_{j2j} = \exp(C \mathbf{x}^T_{j_2} \mathbf{x}_j)$ used in [Pat07] (to simplify I drop the constraint $||x_j||_2 = 1$) the gradient descent step for user i is following:

$$X_{J_i} \mathrel{+}= \ lrate * C * K_{J_i : J_i} X_{J_i}$$

I have not tested gradient descent learning on the entire Netflix dataset, only on a subset of the data (see the end of this section), and this method gave the best result among five methods tested on the same data (see chapter 5 "Experimental results").

The probabilistic model based version of the above methods is multitask learning of Gaussian Processes, that is, we assume that all known and unknown ratings $\mathbf{r}_i$ of a user i are drawn from a multidimensional Gaussian distribution with a covariance matrix K:

$$\mathbf{r}_i^{a-priori} \sim N(\mu + c_i + \mathbf{d} + \mathbf{m},\ K)$$

The posterior distribution of all ratings after observing data is following:

$$\mathbf{r}_i \sim N(\mu + c_i + \mathbf{d} + \mathbf{m} + K_{:,J_i} K^{-1}_{J_i J_i} (\mathbf{y}_i - \mathbf{m}_i),\ K - K_{:,J_i} K_{J_i J_i} K_{J_i,:}) \tag{42}$$

where $\mathbf{y}_i = \mathbf{r}_{J_i} - \mu - c_i - \mathbf{d}_{J_i}$. Note that, independently of the chosen kernel K, this method fits the observed data exactly.

Another possibility is adding noise on the diagonal:

$$\mathbf{r}_i^{a-priori} \sim N(\mu + c_i + \mathbf{d} + \mathbf{m},\ K + \sigma^2 I)$$

Adding noise is justified by experiments with re-rating items [Ama09], which show that a user often rates the same item differently, hence we do not observe the data exactly. The posterior distribution is following:

$$\mathbf{r}_i \sim N(\mu + c_i + \mathbf{d} + \mathbf{m} + K_{:,J_i} (K_{J_i J_i} + \sigma^2 I_{J_i})^{-1} (\mathbf{y}_i - \mathbf{m}_i),\ K + \sigma^2 I - K_{:,J_i} (K_{J_i J_i} + \sigma^2 I_{J_i})^{-1} K_{J_i,:})$$

The kernel K can be learned in the multitask learning framework [Car97, Bak03] with one task per each user. A simplification made here is treating observations as randomly missing, ignoring the specific structure of missing data. Several variants of multitask Gaussian Processes models learned by expectation-maximization have been proposed [Sch05, Yu05b, Yu07a, Yu07b, Yu09a, Yu09b]. It turned out, that for the Netflix Prize data with 480,189 users, the maximum likelihood step in EM methods gives good enough results, even without regularization and without dimensionality reduction. I will describe one variant of EM learned GP multitask model, the NPCA method [Yu09a], which had very good accuracy on the Netflix Prize data.

NPCA [Yu09a] learns the hyperparameters mean $\mathbf{m}$ and covariance K with expectation-maximization. In the expectation step for each user the posterior distribution (42) is calculated, with $\mathbf{m}$ and K fixed. In the maximization step new K and $\mathbf{m}$ are calculated as follows:

$$K_{new} = \frac{1}{N}\sum_{i=1}^{N}(E\mathbf{y}_i E\mathbf{y}_i^T + Cov\ \mathbf{y}_i) =$$

$$= K + \frac{1}{N}\sum_{i=1}^{N} K_{:,J_i}\Big(K_{J_i,J_i}^{-1}(\mathbf{y}_{J_i} - \mathbf{m}_{J_i})(\mathbf{y}_{J_i} - \mathbf{m}_{J_i})^T K_{J_i,J_i}^{-1} - K_{J_i,J_i}\Big)K_{J_i,:} \tag{43}$$

$$\mathbf{m}_{new} = \mathbf{m} + \frac{1}{N}\sum_{i=1}^{N} E\mathbf{y}_i = \mathbf{m} + \frac{1}{N}\sum_{i=1}^{N} K_{:,J_i}K_{J_i}^{-1}(\mathbf{y}_{J_i} - \mathbf{m}_{J_i})$$

After moving K out of both sides of the formula (43) to reduce computational complexity, we get an algorithm called Fast NPCA [Yu09a]:

```
for iter in 1..niter
    for i in 1..N
        z = K_{J_i}^{-1}(y_{J_i} - m_{J_i})
        b_{J_i} += z
        B_{J_i,J_i} += zz^T - K_{J_i}^{-1}
    K += (1/N) KBK
    m += (1/N) Kb
```

We can note, that in NPCA the posterior variance at the points of observed data is always equal to zero (the algorithm fits the data exactly), independently of the choice of kernel K. Because 99% of data is missing, it has little importance for the estimation of K by maximum likelihood (the algorithm averages over predictions for all data, observed and missing),

Experiments with nonparametric estimation of single entries in the covariance matrix K, with a similar method to the used in sections 4.2.2, 4.2.3 and 4.2.5, suggest that the prior distribution on entries of K (if, to simplify, we assume that the entries are independent), is similar to a Laplace distribution, or normal product distribution, and not to a Gaussian distribution. It suggests to estimate the entries of covariance matrices with L^1 regularization, but I have not tested that possibility (also if we assume that the multiplied ratings used for estimation are Gaussians, we should change the likelihood function accordingly).

Related to NPCA are the algorithms BSRM [Zhu08] and NREM [Yu09b]. BSRM bases on the Stochastic Relational Model (SRM) [Yu06a], a Gaussian Processes model, where the covariance matrix is defined between the pairs (user,item):

$$Z \sim N(0, \Sigma\Omega)$$

$$y_{ij} \sim N(z_{ij}, \sigma^2)$$

hence $Cov(z_{i_2j_2}, z_{ij}) = \Sigma_{i_2i}\Omega_{j_2j}$, where the matrices Σ and Ω have the prior distribution of inverse-Wishart Process [Zhu08]: $\Sigma \sim IW_\infty(\delta, \Sigma_0)$, $\Omega \sim IW_\infty(\delta, \Omega_0)$. BSRM [Zhu08] sa a version of SRM with dimensionality reduction. A similar method to NPCA and BSRM is NREM [Yu09b] (Nonparametric Random Effects Model). For both BSRM and NREM the following use of the missing data structure was proposed – using as additional item features the feature vectors from PCA of the binary indicator matrix of data observed (see the method BSRM/F in [Zhu08] and NREM-2 in [Yu09b]). The reader is directed to the works [Zhu08, Yu09b] for detailed descriptions of the methods BSRM and NREM.

My implementations of Fact NPCA and NREM with additional features taken from regularized SVD confirmed the good individual accuracy, but did not improve the ensemble and were not included in the final ensemble listed in chapter 5. Similarly, earlier I implemented multitask GP [Yu07b, Yu05a] initialized to the SVD-based kernel used in KRR [Pat07], and it was the most accurate method among all methods implemented by me at that time, but it did not improve the ensemble accuracy.

The described kernel methods modelled $\mathbf{m}$ and K by maximum likelihood point estimation inside an EM algorithm, assuming identical $\mathbf{m}$ and K for all users, except for modelling the time (modelling $\mathbf{m}|\mathbf{t}_i$ and $K|\mathbf{t}_i$ in most KRR variants). A subject for future research is how to properly make use of the structure of missing data, for example, by modelling $\mathbf{m}|J_i$ and $K|J_i$. An untested method is to use dimensionality reduction similar to NSVD:

$$\min_{W,Z} \sum_{ij \in Tr, j_2 \in J_i} (y_{ij} y_{ij_2} - |J_i|^{-0.5} \sum_{k=1}^{K} z_{jk} z_{j_2 k} \sum_{j_3 \in J_i} w_{j_3 k})$$

Another unexplored possibility is predicting both ratings and indicators of observed data with a hybrid of kernel ridge regression and kernel logistic regression. KRR may be well suited to model the decrease of expected rating for non-user-selected data, for example by using appropriately defined distance from the group of items rated by the user.

Table 34 lists results of different kernel methods on the Netflix Prize dataset.

Table 34: Multitask kernel methods.

Method	Features	RMSE_{15}	RMSE_{quiz}
BSRM/F [Zhu08]	200+30		0.8880
KRR ext. poly on NSVD1 [Tos09]			0.8971
NPCA [Yu09a]	∞		0.8926
NREM-2 [Yu09b]	∞, 20		0.8853
SVDKRRG [Pat07]	96	0.9023	($\sim$ 0.9050)
SVDKRRG+BASIC [Pat07]	96	0.9006	($\sim$ 0.9035)
SVDKRRG2	96	0.9038	($\sim$ 0.9070)
KRRT	96	0.8956	($\sim$ 0.8985)
KRRT2	96	0.9040	($\sim$ 0.9070)
KRRS2	100	0.8849	($\sim$ 0.8880)
KRR3	100+100	0.8846	($\sim$ 0.8875)
SVDKCOV	30+70	0.8844	($\sim$ 0.8875)

To speed up calculations, in the methods SVDKRRG, SVDKRRG2, KRRT, KRRT2 the number of observations was limited to 500 most frequently rated movies, and in KRRS2, KRR3 to 700 movies. SVDKRR+BASIC was a result of a linear regression of KRR predictor with six predictors BASIC (five per-user empirical probabilities of ratings, and the regularized movie average after subtracting the user mean from ratings).

Two methods, NPCA (without learning the mean $\mathbf{m}$) and KRRG, were tested on a small subset of the Netflix Prize data with 10,059 users and 500 movies (see section 3.3 for description how the small dataset was chosen). KRRG was KRR with a Gaussian kernel learned by gradient descent. The results are shown in table 35. NPCA gave much better results on the entire dataset probably because the maximum likelihood step works better when it averages over more users (tasks).

The kernels used in the above methods typically had the form $k_{j2j} = g(\mathbf{x}_{j_2}^T \mathbf{x}_j)$. This form appears also often generally in the kernel methods literature, in other tasks than collaborative filtering. After choosing the weighting w_k of the input features $x_{jk} = w_k v_{jk}$, remains the choice the monotonic g function. For KRR a small change in the shape of g causes a large change of the results of the algorithm. In comparison, for K-NN methods

Table 35: Multitask kernel methods. Small dataset.

Method	K	RMSE_{small}
KRRG [Law09]	100	0.7415
NPCA	∞	0.7520

the choice of g does not influence the resulting predictions by much (does not change the weighting by similarity by much, and the nearest neighbors are the same independently of g).

Multitask Gaussian Processes can also be used with an ordinal output, as it was done in the hierarchical Bayesian framework in [Yu06b].

4.6 Other methods

Ensembles developed for the Netflix Prize task contained mainly variants of matrix factorization, RBM and distance-based methods (such as K-NN methods and kernel methods). Other kinds of methods rarely improved the ensembles. I will list here a few methods, that did not fit into the classification in previous sections.

An interesting alternative for the factorization methods and distance-based methods is the method of opinion diffusion [Zha07b, Zha07c], inspired by the physical process of heat conduction, with a computational optimization by employing a Green function approach.

Matrix factorizations are close to neural network methods. Neural networks with one and two hidden layers were tried on the Netflix task [Tos08b, Tos09].

Clustering users or movies with K-means can be used for collaborative filtering [Mar04]. The ensemble in [Pat07] contained stacked K-means clustering of users, denoted as KME0AVG and KME1AVG. KME0AVG was an ensemble of averaged 10 runs of user clustering by K-means [Pat07]. KME1AVG was the same K-means ensemble algorithm run on residuals of KME0AVG.

If, instead of discrete membership in K-means clusters, we will allow shared, fuzzy membership, we obtain a nonlinear dimensionality reduction method , for example, [Wu08] proposed the method Modified Fuzzy C-means (related to nonlinear factorizations – section 4.3.8).

Table 36 lists the RMSE's of a few methods that did not fit into the classification in previous sections.

Table 36: Other methods.

Method	Features / Clusters	RMSE_{15}	RMSE_{quiz}
Neural Network, 1-layer [Tos08b]	50		0.9041
Neural Network, 2-layer [Tos08b]	10+10		0.9300
MFCM1 (Modified Fuzzy C-means) [Wu08]	40		0.9110
KME0AVG + BASIC [Pat07]	4-24	0.9410	($\sim$ 0.9440)

I have described so far in this chapter mainly the methods that contributed to ensembles. A large majority of the methods implemented by me did not improve the ensemble accuracy (about 90% of methods in my experience – experience of other teams was similar), and perhaps it is worth to tell a bit more about that part of the universe of methods, that was explored, but turned out to be not useful. Most of these methods were different modifications of regularized SVD and KRR. A few other attempts not improving my ensemble were, for example: multilayer neural networks, such as 3-layer and 4-layer autoencoders (in some tries, initialized by RBM, as in deep networks [Sal09]); different

ways of postprocessing SVD features, such as random forests method, different kinds of regression, KRR with kernels different from Gaussian; per-user regression using features learned by different methods; "best movie decomposition", where instead of learning user preferences one most preferred movie was picked; an NN-like version of RBM, or variants of SVD with clustered users or movies.

4.7 Using item metadata

In the Netflix Prize dataset movies were described only by the title and the year of production. One seemingly obvious idea to improve accuracy was to augment the Netflix Prize dataset with additional, publicly available data about movies, such as genre, actors, writer, director, language, etc. and to try hybrid collaborative- and content-based predictions. A counter-intuitive result of many teams' experiments [Lee08, NF07, Pil09b] was that the additional movie metadata from IMDb or Wikipedia does not improve predictive accuracy.

Similarly, I also tried to use the additional data, obtaining no significant improvement. I correlated subsets of the Netflix data with IMDb and with Wikipedia infoboxes, using a snapshot of the two data sources from September 2007. The movies and TV series in the Netflix Prize dataset were automatically matched with the entries in the IMDb and Wikipedia, using a heuristic distance based on the longest common substring and edit distance between titles, and on the difference between the production years. This method, among the 17770 titles from the Netflix database, found about $70-80\%$ of the titles in the IMDb database, and found about 30% of the Netflix titles in the Wikipedia infoboxes. The matching ratio can be improved using newer data and a more accurate matching method. In the work [Lee08] about 77% among 15k Netflix titles were matched with Wikipedia infoboxes, using a snapshot of Wikipedia from February 2008. For the 2000 most popular movies and TV series in the Netflix database, used in the application described in section 6.2.3, with a lot of hand edition I obtained a near-perfect match with the IMDb database from the year 2010.

In an attempt to improve prediction I augmented the Netflix Prize data with a set of features taken from IMDb (snapshot from September 2007). Two sets of 200 most frequent features were extracted for about 66% of movies from the Netflix dataset, for which there was a large confidence that they are correctly matched with IMDb. The first set of 200 features included: genres, keywords, countries, language, locations. The second set of 200 features included: countries, actors, actresses, writers, composers, directors. Table 37 shows the most frequent 20 features from the first set.

The general result of augmentation was negative. IMDb features improved only the simplest models that do not use the variables learned from SVD. I will describe shortly one of the experiments. The set of IMDb features was used to heuristically construct a similarity measure between movies, and this similarity was used in several versions of Kernel Ridge Regression, on top of residuals of biases from the model SVD with biases. KRR with the IMDb-based heuristic similarity had close accuracy ($\text{RMSE}_{15} \approx 0.97$) to KRR with the time similarity alone: $C * \log |t_{ij} - t_{ij_2}|$ (without using the SVD features – unlike the KRR methods from section 4.5.3). Using as the kernel the sum of the IMDb similarity and the time similarity improved RMSE_{15} to .9605. IMDb features carry information capable of improving the simplest models, but attempts to merge the IMDb similarity with the full KRR method (based on the SVD features) did not improve the accuracy of the method, nor the accuracy in combination with the ensemble.

In [Pil09b] a movie-oriented adaptation of NSVD1 [Pat07] was proposed to predict ratings using item metadata, and the resulting method was better in the experiments comparing to other tried methods. [Pil09b] attempted to quantify the amount of information contained in ratings, and a model of biases trained on a subset of data with as low as 10 ratings per movie gave better RMSE than modelling the metadata. Because, in

Table 37: Metadata – the most frequent 20 features.

	Feature	Count	Movie IDs
1.	English	10364	0 4 7 8 9 11 12 14 15 16 17 18 20 21 23 25 27 28 29 ...
2.	USA	8853	0 2 4 7 8 9 11 12 14 15 17 18 20 21 25 27 28 29 30 ...
3.	Drama	4743	7 15 17 18 19 23 25 27 28 29 35 41 43 46 50 54 55 ...
4.	Comedy	3471	7 8 11 19 21 27 29 50 53 64 67 72 77 83 94 110 116 ...
5.	independent-film	2826	8 14 15 17 21 23 30 41 51 65 66 74 79 88 106 109 ...
6.	character-name-in-title	1886	27 30 31 35 45 50 56 60 70 72 93 94 112 117 160 164 ...
7.	Thriller	1881	15 16 23 25 40 54 79 82 88 92 104 107 108 117 121 ...
8.	Action	1721	12 15 16 25 47 54 57 65 68 76 77 83 88 90 108 117 ...
9.	Romance	1652	11 17 19 21 29 35 49 53 62 94 116 147 155 160 163 ...
10.	based-on-novel	1603	12 25 35 44 55 76 94 151 196 209 211 251 256 273 ...
11.	murder	1486	11 12 17 23 25 54 55 57 121 124 149 174 185 196 ...
12.	UK	1472	16 17 35 56 63 109 112 161 186 204 219 223 228 ...
13.	Los Angeles, CA, USA	1323	21 25 29 64 77 79 106 107 108 121 126 129 136 154 ...
14.	Crime	1186	16 25 54 55 107 108 122 126 136 146 167 174 185 ...
15.	Documentary	1167	0 4 7 9 14 30 31 51 60 70 93 95 105 112 118 164 175 ...
16.	female-nudity	1122	8 17 29 51 65 106 109 136 150 167 196 203 204 282 ...
17.	Adventure	1093	12 15 19 27 45 47 50 57 65 76 77 83 88 117 240 243 ...
18.	Horror	1071	8 15 23 40 66 92 121 128 130 150 171 187 196 209 ...
19.	Family	983	0 19 27 34 45 47 50 67 72 77 83 151 154 238 251 254 ...
20.	Sci-Fi	905	15 27 40 47 67 76 121 130 188 208 215 216 243 275 ...
...	...	...	...

comparison with other types of items, movies have relatively much meaningful metadata available, it can be presumed, that the conclusion about small number of ratings being better than metadata generalizes, and holds in situations of evaluation and recommendation of items other than movies. Metadata can be more useful for accurately predicting the missing data structure (see the KDD'11 task, track 2 [Dro11]), or for adjusting rating prediction for the missing data structure.

The Netflix Prize data contains movies with a lot of ratings – only 2 movies in the Netflix Prize dataset have less than 10 ratings. The real situation in recommender systems is usually different, with a "long tail" of items with very few ratings. To overcome the cold start problem for items, and be able to efficiently recommend rarely rated items or new items without ratings, useful are content-augmented predictions. Several differing approaches have been proposed how to do it. Metadata can be used in a hierarchical probabilistic model, where a shared prior for user weights is learned with an EM algorithm [Zha07a]. In a related approach, metadata is treated as fixed vectors of item features in a matrix factorization [Tak07a] – the probabilistic versions of this concept are: Matchbox [Ste09], BMFSI [Por10a], RLFM [Aga09], and a specialized model, fLDA, that models topics, when item features are words [Aga10]. Another possibility is the already mentioned neural-network-type algorithm, movie-oriented NSVD1, adapted to model metadata [Pil09b]. Yet another way is predicting ratings directly, using linear regression with metadata as predictors, with different sets of weights learned for different clusters of users [Kag09]. Feature-Based Matrix Factorization [Che11b] unifies using internal and external features, encompassing, for example, biases, SVD++, neighborhood information and time effects. Bayesian Factorization Machines [Fre11] are a similar, more general framework. [Hid12] factorizes item metadata, and uses this factorized representation to extend a matrix factorization.

In this work a simple method of using metadata for content-based augmentation of SVD was used (see sections 4.3.4 and 6.2.3). IMDb features are used to predict the SVD item features using ridge regression, and the predictions of the regression are used as priors

for the SVD item features. This method was used to annotate and understand better the automatically learned SVD features (section 4.3.4), and was also used for purely content-based recommendations for movies rated after 2005 (where no ratings are available), added to the recommender system described in 6.2.3.

A different simple heuristic method of using metadata was used in the SVD-based recommender system described in section 6.1: the heuristic was to add artificial users who like the given feature or a set of features. The advantage of artificial users is that they can be used to improve an already implemented collaborative filtering algorithm (for example, regularized SVD) without modifying it. The accuracy of this method was not evaluated, but it works satisfactorily well in practice, fulfilling its goal of helping in cold start situations. An analogous technique of creating artificial items can be tried out, with the artificial item ratings (or otherwise expressed artificial preferences) dependent on provided user metadata.

4.8 Combining models

In the process of searching for methods with the best predictive accuracy it is often observed that various ways of combining methods improve accuracy. Usually such a situation occurs, when two very different methods, thus capturing different aspects of the data, both have good accuracy.

I will describe several ways of combining methods, that proved to be effective in the experiments of many teams working on the Netflix Prize task: postprocessing, integrating, or blending methods.

Preprocessing and postprocessing the methods by one another (called also stacking) are convenient ways of exploring possible combinations, because they do not require to change the learning algorithm (sometimes they need only tuning several parameters).

Good combinations (improving accuracy) discovered by stacking or blending can be usually further improved by creating integrated models.

A useful technique was blending, that is combining many methods into an "ensemble". In the Netflix task most often simple linear regression, and its regularized version, ridge regression, were used for blending, but some alternatives for linear blending also performed well.

Real-world machine learning applications are a compromise between the predictive accuracy and the simplicity of a method. Usually in real-world applications it is enough to use only one model, because the relatively small accuracy gain from combining methods rarely outweighs difficulties coming from the additional complexity. Maintaining and successively improving an ensemble of methods is primarily a good methodology of finding out the most accurate methods, increasingly understanding which process generated the data (or rather, increasingly understanding the aspects of that process, that are important for the chosen prediction task, discovering the right discriminative effects, etc.). It is an effective way, in terms of saving the analyst's time and effort, of exploring the space of possible methods.

4.8.1 Preprocessing and postprocessing

While developing solutions for the Netflix Prize many teams observed that the accuracy of methods can often be improved by stacking methods, that is running one method on residuals of one another, instead of on the raw ratings. Some combinations gave better accuracy than other ones. In the most accurate stacked models visible is a hierarchy, named in [Bel07a] as modelling at different scales: in those best models we observe the transition from modelling global effects, through the most efficient, middle-level dimensionality reduction layer, to modelling local, item-item effects.

In this work as a preprocessing step most commonly were used different kinds of global effects [Fun06, Bel07b, Pot08] (see section 4.3.1), and sometimes the NSVD term taken from SVD++ variants. The middle-level layer were dimensionality reduction methods, such as regularized SVD, SVD++, conditional RBM. On the Netflix data, this layer has the largest modelling capability, and is very accurate even without the remaining, few-parameter and highly-parameterized layers. As a postprocessing step well worked neigborhood modelling methods, such as K-NN [Bel07b, Tos08b, Pio09] or per-item linear models, capturing large local item-item correlations. The method KRR/GP [Pat07] with a Gaussian kernel has properties of the last two layers.

A useful observation is that parameter tuning is easy in the methods used in the last, postprocessing layer, because there is no need to re-learn the remaining methods in the stack.

Table 38 lists selected accurate combinations of methods by postprocessing. Some other combinations were listed in table 31 section 4.5.1, and in table 29 in section 4.4.1.

RBM + KNN was the most accurate stacked method in the ensemble [Bel07c]: RBM [Sal07a] was combined with a variant of K-NN with jointly learned weights [Bel07b]. NNMF + KNN [Bel07c] was non-negative matrix factorization postprocessed by the same KNN method [Bel07b].

Neighborhood-aware MF [Tos09, Tos08b, Tos08a] is a weighted combination of matrix factorization item-item K-NN model, and user-user K-NN, with inserting predictions for the qualifying set as additional ratings.

HYBRID1 with NB correction S1 [Tak08c] combines MF with NSVD1, and postprocesses the result with a neighborhood-based correction using cosine similarity between the learned item feature vectors from MF.

The method SBRAMF-* + KNNMovieV3-2 is SBRAMF-UTB-UTF-MTF-ATF-MFF-AFF [Tos09] (briefly described in section 4.3.6 "Time effects"), postprocessed by KNNMovieV3-2 [Tos09] (a K-NN method similar to KNNMovieV3 [Tos08b]; see section 4.5.1 'K-nearest neighbors").

The method bk4-f200z4-nlpp1-knn3-1 [Pio09] is the integrated model bk4-f200z4 [Pio09] (briefly described in the next section), with a nonlinear output transformation, and postprocessed by a K-NN variant with jointly derived weights [Bel07b], where the neighbors were chosen so that the similarity does not exceed a fixed threshold. With $\mathrm{RMSE}_{quiz} = 0.8713$ it was the most accurate method for the Netflix Prize task among the methods mentioned in this work. For a more detailed description of that method the reader is directed to the paper [Pio09].

At the bottom of table 38 are listed my implementations. DRBMKNN is DRBM, described in section 4.4.1 "Restricted Boltzmann Machines", postprocessed by modified KNNMovieV3 [Tos09], described in section 4.5.1. SVDMT2KNN is SVDMT2, described in section 4.3.6 "Time effects", postprocessed, as the earlier method, by the modified KNNMovieV3. SVDMT2KNN had the best accuracy among all single methods implemented by me (if we count stacked methods as single methods, and e.g. combining by blending as separate methods).

Stacking has the advantage that the combined methods are implemented once, and do not need to be modified. With relatively little effort multiple combinations can be tested (see, for example, the final solutions [Pio09, Tos09, Kor09b]). It is possible to automate to some degree the process of searching for good combinations, for example, by checking N^2 possible stackings of any two methods among the N methods implemented, but I have not tried this option.

Table 38: Stacked methods.

Method	Features	K (NN)	$RMSE_{15}$	$RMSE_{quiz}$
NNMF + KNN [Bel07c]	30	180		0.8953
CRBM + KNN [Bel07c]	100	50		0.8888
CRBM + KNN + KNNMovieV3 [Tos08b]	150	55+122		0.8832
Neighborhood-aware MF [Tos09, Tos08b, Tos08a]	100	50		0.8856
HYBRID1 with NB correction S1 [Tak08c]	400	40		0.8845
SBRAMF-* + KNNMovieV3-2 [Tos09]	150	141		0.8758
bk4-f200z4-nlpp1-knn3-1 [Pio09]	200	≤ 60		0.8713
DRBMKNN	100	50	0.8888	($\sim$ 8920)
SVDMT2KNN	30	50	0.8819	($\sim$ 8850)

4.8.2 Integrated models

A discovered good combination of two methods (with better accuracy than any of the two methods) indicates the possibility to create an integrated method with even better accuracy. Good combinations to integrate were usually discovered by stacking methods (the previous section 4.8.1), or linear blending (the next section 4.8.3). Having a well performing combination of two or more models, the data analyst is faced with a riddle, what is the right way to integrate the models. The models appearing in the Netflix Prize task were usually integrated by modelling the rating with a sum of outputs of the two models, with jointly learning parameters of both models. Sometimes a more complex way of integration was needed, such as merging parts of both models.

Complex integrated models had best obtained accuracy on the Netflix Prize task. Integrating discovered methods is a part of the process of approaching the "optimal prediction method" that stems from the unknown model that generated the data.

Besides the predictive accuracy, a focus of this work is also reflecting, how to save analyst's time and effort in the engineering process of developing accurate methods. Looking at the development of methods of since releasing the data by Netflix in 2006, several paths of discovery can be noticed. It is likely that the process of discovering accurate methods will be similar when working on other prediction tasks. A question is how to shorten and make easier the process of discovery, and how to automate parts of it.

One path of discovery was finding out the appropriate use of user and movie biases in dimensionality reduction methods. The standard dense SVD from linear algebra does not use biases, and the PCA method uses only non-regularized column centering. It was noticed, that preprocessing regularized SVD by removing both column and row biases, using regularized estimates, improves accuracy [Fun06]. Integrating both biases as part of the SVD model, and learning them together with the remaining parameters, further improved accuracy [Pat07, Tak07a, Tak07b]. A variety of well performing modifications of biases, and other global effects were found (see section 4.3.1 "Global effects").

Another path of discovery to distinguish was adding temporal effects to the models: integration of global time effects [Bel07b], and the noticed short-term correlations of residuals of user ratings and correlations of user preferences (see section 4.3.6 "Time effects"). The per-day change in user biases [Tom07, Pot08, Kor08] was included in the models as an additional term directly predicting the output rating. The per-day change in user preferences was modelled as an additional term added to each user preference parameter [Kor08].

An important combination was integrating regularized SVD [Fun06] with NSVD [Pat07], which gave the method SVD++ [Bel07e] (see section 4.3.5). Here the integration was more complex – based on merging the movie parameters, and summing the parts parameterizing user preferences in both models, obtaining an analogous algorithm to Conditional RBM

[Sal07a].

Another combination that significantly improved accuracy was merging regularized SVD with neighborhood methods, like K-NN. Here the best results were obtained with per-item linear models [Kor08] (see section 2.2), which can be understood as K-NN with jointly learned weights, without the normalizing sum in the denominator, and without the limit on the number of nearest neighbors. SVD methods with integrated neighborhood components are an example of learning at multiple scales, where the additional scale of neighborhood allows to model many strong item-item relations, which, because of their amount and limited scope, cannot be captured by dimensionality reduction methods like SVD and RBM. Other developed methods that modified the factorization model by modelling item-item neighborhood were: matrix factorization with user factors augmented by item-item similarity [Bel07a], and applying kernel methods like KRR [Pat07] (see section 4.5.3).

The most accurate models discovered for the Netflix data [Pio09, Kor09b, Tos09] integrated multiple methods and many effects that improve accuracy (various global effects, the use of implicit information, modelling neighborhood, and temporal effects, including frequency effects). Some of the most complex integrated models were already described in sections 4.3.5 "Improved user preferences" and 4.3.6 "Time effects". Table 39 compares RMSE of selected models of different authors.

"SVD++ integrated with NB" [Kor08] integrates SVD++ [Bel07e] (section 4.3.5) with the global neighborhood term [Kor08] (per-item linear models, section 4.5.2). "Time-aware SVD++ integrated with NB" [Bel08] further adds time-changing user biases, movie biases, and user preferences.

Among the published models, a single model that obtained best accuracy in the Netflix Prize was BK4 [Pio09], that is the mentioned above time-aware SVD++ integrated with the global neighborhood term [Bel08], extended by using additional time effects: factorization of biases vs. time, time- and frequency-dependent movie features, movie biases, implicit feedback term, neighborhood term, and per-user scaling term. Table 39 lists a variant of BK4 named bk4-f200z4 [Pio09]. I quote the general form of the BK4 model in the following listing to give the reader an idea of its complexity level. For a detailed description of the parameters the reader is referred to [Pio09].

$$
\begin{aligned}
z(u,m,t) &= \mu + b_u(u,t) + b_m(m,t)\, s(u,t) \\
&+ \sum_{l}^{n} q_l(m,t) \left[p_l(u,t) + \frac{s(u,t)}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_l(j,t) \right] \\
&+ \sum_{j \in R^k(m;u)} w_w(m,j)\,(r(u,j) - baseline_2(u,j))\, w_{m,j} \\
&+ \sum_{j \in N^k(m;u)} w_c(m,j)\, s(u,t)\, c_{m,j}
\end{aligned}
$$

$$
b_u(u,t) = b_{u0}(u) + \sum_{l}^{n_{bu}} b_{u1,l}(u) \left[b_{uf,l}(f) + b_{ut,l}(t_u) \right] + b_{u2}(u,t)
$$

$$
b_m(m,t) = b_{m0}(m) + \sum_{l}^{n_{bm}} b_{m1,l}(m) \left[b_{mf,l}(f) + b_{mt,l}(t_m) \right]
$$

$$
s(u,t) = 1 + s_0(u) + \sum_{l}^{n_{su}} s_{1,l}(u) \left[s_{f,l}(f) + s_{t,l}(t_u) \right] + s_2(u,t)
$$

$$
q(m,t) = q_0(m) + q_1(m) \left[\sigma\left(f\alpha_q + \beta_q \right) + q_t(m, t_{36}) + q_f(m, f_{36}) \right]
$$

$$
p(u,t) = p_0(u) + p_1(u) \left[\sigma\left(t_u \alpha_p + \beta_p \right) + p_2(u,t) \right]
$$

$$
y(j,t) = y_0(j) + y_1(j) \left[\sigma\left(f\alpha_y + \beta_y \right) + y_t(j, t_{36}) + y_f(j, f_{36}) \right]
$$

$$
\sigma(x) = \frac{1}{1 + e^{-x}}
$$

$$
w_w(m,j) = \frac{\frac{1}{1 + \beta_w t_\Delta(m,j)}}{\sqrt{\sum_{l \in R^k(m;u)} \left(\frac{1}{1 + \beta_w t_\Delta(m,l)} \right)^2}}
$$

$$
w_c(m,j) = \frac{\frac{1}{1 + \beta_w t_\Delta(m,j)}}{\sqrt{\sum_{l \in N^k(m;u)} \left(\frac{1}{1 + \beta_w t_\Delta(m,l)} \right)^2}}
$$

$$
\hat{r}(u,m,t) = \begin{cases} z(u,m,t) & linear\ model \\ \sigma_2(z(u,m,t)) & non-linear\ model \\ \sigma_3(z(u,m,t)) & non-linear\ model \\ \sigma_4(z(u,m,t)) & non-linear\ model \end{cases}
$$

Much simpler than the above model BK4, while having close accuracy, is the model PQ2 [Kor09b], briefly described in section 4.3.6.

Table 39: Integrated models.

Method	Features	K (NN)	$RMSE_{quiz}$
SVD++ integrated with NB [Kor08]	200	∞	0.8868
Time-aware SVD++ integrated with NB [Bel08]	200	300	0.8789
bk4-f200z4 [Pio09]	200	≤ 445	0.8760

Finally, let's remark that there were accurate combinations of methods, for which it is unclear, how to create a proper integrated model, for example, RBM postprocessed by K-NN [Bel07c].

4.8.3 Blending predictions

A framework that turned out to perform exceptionally well in tasks of optimizing predictive accuracy is blending outputs of many independently trained methods. Blending gives best results when combining possibly accurate, but diverse methods. In the Netflix Prize task, the most frequently used blending method was linear regression (usually regularized) of predictions on a hold-out set. Besides leading to accurate prediction, an advantage of

maintaining an ensemble of diverse methods was the possibility to evaluate how different methods combine with each other, what was useful in exploring the space of possible models, and gave insights, in which direction to further develop the algorithms.

The methods to be blended were learned on the training set with a chosen hold-out test set excluded. Each method produced one or more predictors (sets of predictors) for the test set (ratings unused in training), and the resulting predictors were blended to predict ratings on the test set, with the goal to obtain best generalization accuracy on the additional validation set (RMSE_{quiz} or RMSE_{test}).

The Netflix Prize dataset and various choices of the hold-out test set were discussed in sections 3.3 and 3.4. One way was using the entire probe set as the hold-out set, but a disadvantage of that choice was the necessity to retrain methods on the entire training set, after calculating the regression weights. A compromise between accuracy and convenience was using only a portion of the probe set as hold-out, so that the methods needed only to be trained once. For example, in [Pat07] and in this work 15% of probe is used as hold-out, and in [Tak08a] 10% of probe was used, chosen so that $\text{RMSE}_{10} \approx \text{RMSE}_{quiz}$.

In prediction contests such as the Netflix Prize, typically an even more efficient method of hold-out blending is possible. In [Kor09b, Tos09] a method of using quiz (validation) set for blending was proposed, which allowed to include the entire probe set in one-time training of methods. The method is all the more interesting because for the quiz set no ratings were available, only the values of RMSE_{quiz} reported by the Netflix Prize automatic evaluation system. The method in [Kor09b, Tos09] was to reformulate the task of regression, so that the resulting regression coefficients depend only on RMSE_{quiz} and on the sufficient statistics not containing the response variable (ratings). Let X be the matrix of predictors and let $\mathbf{y}$ be the predicted response vector. To make predictions $\hat{y}_{new} = \mathbf{x}_{new}^T \hat{\boldsymbol{\beta}}$ we need to calculate $\hat{\boldsymbol{\beta}}$:

$$\hat{\boldsymbol{\beta}} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}$$

The terms containing $\mathbf{y}$ are:

$$c_i = \sum_j x_{ij} y_j$$

Because $ab = \frac{1}{2}(a^2 + b^2 - (a-b)^2)$, we can write:

$$c_i = \frac{1}{2}\Big(\sum_j x_{ij}^2 + \sum_j y_j^2 - \sum_j (x_{ij} - y_j)^2 \Big)$$

We see that $\hat{\boldsymbol{\beta}}$ can be calculated using MSE values of individual predictors on the validation set with unknown response values. The remaining statistics needed depend only on the predictors, which values we have.

In my experiments I used blending on 15% of the probe set, with methods trained once on the training set with the 15% of the probe set excluded. I have not used all produced predictors for blending – feature selection was necessary. Not all produced predictors improve the ensemble accuracy, and using a large number of noisy, unnecessary predictors can significantly decrease the accuracy. Considerations about model selection for linear models give criteria as AIC, BIC (see section 2.4), which suggest modifying the likelihood of the observed data by a penalty linearly growing with each additional predictor. For a predictor to be useful, its contribution to explaining variance (or to increase the likelihood of data) must outweigh a certain threshold, which is to be determined, for example, by cross-validation (with a remark, that one fixed threshold for all predictors is a simplification – it should depend e.g. on correlation with other predictors).

Feature selection was especially needed after extending the linear regression by multiplications of predictors (called two-way interactions). In this work's ensemble 69 predictors and 81 two-way interactions were used (in comparison to the previous ensemble [Pat07]

with 56 predictors and 63 two-way interactions). The two-way interactions for the ensemble were chosen using two criteria: statistical significance in the linear regression, and the drop of the sum of squared errors. The selected interactions with their contribution are listed in chapter 5. In a similar way, in [Pio09] 5 multiplicative interactions selected by forward feature selection were added to their ensemble.

Feature-weighted linear stacking (FWLS) [Sil09] can be understood as ridge regression with two-way interactions. FWLS added to the ridge regression multiplications of all predictors with a chosen set of 25 meta-features such as the logarithm of movie support, the average number of movie ratings for the movies rated by the user, or standard deviation of SVD prediction. As a result, the accuracy improved from $\text{RMSE}_{test} = 0.863377$ to $\text{RMSE}_{test} = 0.861405$ – a similar order of improvement to the observed in the experiments in this work (chapter 5) after adding 81 two-way interactions to the linear regression.

For the predictors in my ensemble, linear regression without regularization gave very close accuracy to using a small amount of regularization (perhaps it is an exceptional situation because of large collinearity between some predictors). Using regularization has the advantage that it makes possible to use a lot more interactions (even with no feature selection, like in FWLS [Sil09]) without causing large overfitting.

The number of possible two-way interactions is large, and we could try to model them using fewer parameters than one parameter per interaction. In a similar fashion to Factorization Machines [Fre11], we could add to the linear model terms modelling interactions between groups of variables: $(\sum_{i=1}^{P1} \alpha_i x_{k_i})(\sum_{j=1}^{P2} \gamma_j x_{l_j})$. I have not tried this option.

It turned out that well chosen nonlinear blending methods improve accuracy. Very good accuracy was obtained by neural networks with a small number (10-30) of hidden variables. In [Tos08b] neural networks blending improved RMSE by 0.0020 in comparison with linear blending. A similar method to NN blending [Tos08b] was used in [Pio09]. In [Tos09] NN blending was extended to using two hidden layers.

[Tos09] further proposed Ensemble NN blending, which gave best accuracy among all blending methods described in [Tos09]. Ensemble NN blending relied on learning random $k = 4$ predictors, with two additional support-based input predictors $\log(|J_i| + 1)$ and $\log(|I_j| + 1)$. In the best setting, which gave RMSE= 0.8583 in [Tos09], NN with 5 hidden neurons were used, the process of drawing the four predictors was repeated $N = 1060$ times, and the resulting 1060 new predictors were combined by binned ridge regression, on 4 support-based bins.

In [Pio09] a multi-stage classifier was used, that relied on learning several blending methods on residuals of each other. The authors concluded that NN blending [Tos08b, Pio09] was superior to their multi-stage classifier.

Accurate, but with slightly worse accuracy than NN blending, were variants of Gradient Boosted Decision Trees (GBDT) [Kor09b, Tos09] (called also Gradient Boosted Machine) and Bagged GBDT (BGBDT) [Tos09, Jah10]. GBDT can be used for prediction tasks with any loss function, as a black box, with minimal parameter tuning, and it often leads to good accuracy. It makes them a convenient first-choice technique in prediction contests.

Other blending methods used on the Netflix task were: polynomial regression, which adds higher order terms of predictors [Tos09], kernel ridge regression with Gaussian kernel [Tos09], binned linear regression using support-, date-, frequency-, and clustering-based bins [Bel08, Tos09]. In some cases features extracted from SVD, RBM and K-NN [Tos09, Kor09b] were used as additional predictors.

Most individual methods have at least several constant parameters to tune. Instead of tuning those parameters to optimize RMSE of an individual method, the RMSE of the blend can be optimized instead. I have not used this method in this work, but tuning to optimize the blend was used extensively in [Pio09, Tos09] for automatic parameter tuning.

Optimizing blend accuracy was particularly effective for K-NN methods.

Summarizing the subject of blending predictors, linear regression worked well (regularization usually improved accuracy), and among the nonlinear blending methods best results gave neural networks blending, ensemble neural networks blending and decision trees.

In practical applications rarely there is a need to use more than one method, because the cost of complicating the algorithm is rarely outweighted by a relatively small accuracy gain. Nevertheless, the framework of blending, in particular simple regression blending, is convenient to evaluate developed models, and explore the space of possible models in search of the most accurate ones.

"By three methods we may learn wisdom:
first, by reflection, which is noblest;
second, by imitation, which is easiest;
and third by experience, which is the bitterest."

Confucius

5 Experimental Results

This chapter summarizes the results of my experiments on the Netflix Prize dataset. I list the developed ensemble of methods in order of feature selection. I discuss the results, and compare with the results published by others.

All implemented methods were trained once on the training.txt, with 15% of probe set excluded. The 15% of probe set serves as a hold-out set, on which RMSE is calculated (called RMSE_{15}). The tables list RMSE_{15} of the current ensemble part, and, where applicable, RMSE_{15} of individual methods. Other RMSE values appearing in this chapter are RMSE_{quiz}, that is results of validation reported by the automatic evaluation system by Netflix during the contest, and RMSE_{test}, that is the final validation results, made available after the Netflix Prize contest ended. Besides the above, one table gathers values RMSE_{small} from experiments on a small subset of Netflix data, which is 50% dense (as opposed to 1.1% dense whole data). The different types of RMSE were described in more detail (on which hold-out set each RMSE is computed, and what is the corresponding training set) in section 3.4 "Evaluation".

The developed ensemble contained 69 predictors and 81 two-way interactions between them, all blended by linear regression. Only a small part of all implemented variants of methods (less than 10%) gave an improvement of accuracy large enough to add them to the ensemble. Of those included, some methods produced several predictors, but most produced only one predictor.

First I add to the ensemble the global effects that were used as a preprocessing for more complex methods, which will be added to the ensemble at a later stage (including the global effects first will allow us better assess the relative importance of the remaining methods in the ensemble). Table 40 lists 13 predictors coming from global effects, along with cumulative RMSE_{15} after adding each subsequent predictor to the ensemble.

Table 40: Predictors in the final ensemble – global effects.

No	Method	RMSE_{15} combined	Description
1	PME1R	3.17241	user empirical prob. of rating 1, regularized
2	PME2R	2.53228	user empirical prob. of rating 2, regularized
3	PME3R	1.95780	user empirical prob. of rating 3, regularized
4	PME4R	1.48953	user empirical prob. of rating 4, regularized
5	PME5R	1.05960	user empirical prob. of rating 5, regularized
6	MEANMOVR	0.98287	movie mean on residuals of user mean, regularized
7	SVD1C	0.98011	user bias c_i from RSVD2
8	SVD1D	0.97938	movie bias d_j from RSVD2
9	SVD1N9C	0.97854	user bias c_i from SVD1N9
10	SVD1N9D	0.97853	movie bias d_j from SVD1N9
11	SVD1NC	0.97834	user bias c_i from SVD1N
12	SVD1ND	0.97824	movie bias d_j from SVD1N
13	SVD1NUV1	0.95146	first feature $u_{i1}v_{j1}$ from SVD1N

Table 41 lists all remaining methods in the ensemble, added in order of greedy forward feature selection (GFFS). As a next feature is chosen the predictor that results in best RMSE_{15} when combined with all predictors included earlier. The table list the individual accuracy of the method as "RMSE_{15} individual" (where applicable), and the accuracy of the current fraction of the ensemble as "RMSE_{15} combined". A simple implementation of GFFS was used with computational complexity $O(NP^3 + P^5)$, where $N = 211365$ is the number of observations (15% of the probe set), and P is the number of considered features. The complexity of GFFS can be improved, if necessary, to $O(NP^2 + P^4)$ [Pil09a].

Table 41: Predictors in the final ensemble, sorted by greedy feature selection

No	Method	Global Effects	RMSE_{15} individual	RMSE_{15} combined	Description	Described in
14	JT2		0.8798	0.87920	JT's model K=60	4.3.3, [Tom07]
15	KRRS2		0.8849	0.87367	KRRT on ALS SVD++ K=100	4.5.3
16	SVDMT2KNN		0.8819	0.87147	SVDMT + KNN, repeated learning	4.5.1, 4.3.6
17	DRBM		0.9101	0.87072	Directed Conditional RBM K=100	4.4.1, [Sal07a]
18	SVDM		0.8919	0.87002	VB-like SVD++ K=30, LOO day bias	4.3.6
19	QNSVD1		0.9679	0.86958	NSVD1 K=85, probe+qual as implicit	4.3.5
20	DRBMKNN		0.8888	0.86924	Directed CRBM K=100 + KNN K=50	4.4.1, 4.5.1
21	KNN20			0.86885	Sum of residuals of 20 NN, RSVD2 dist.	4.5.1
22	DRBM3		0.9192	0.86856	Directed RBM K=100	4.4.1, [Sal07a]
23	RRBA		1.0129	0.86837	Per-movie linear model, $e_i = 1$	4.5.2
24	log(CMOV)			0.86822	log(movie support)	
25	SVDRR	7-8	0.9088	0.86806	Per-user RR on normalized RSVD2 vec.	4.5.2, [Pat07]
26	KRRT2	7-8	0.9040	0.86783	KRR, linear kernel, time dependent	4.5.3
27	NSVD2A		0.9633	0.86766	NSVD2 K=5, w/o qualifying data	4.3.5
28	JT3		0.8861	0.86753	JT's simplified model K=60	4.3.3, [Tom07]
29	KNN1			0.86745	Residual of 1-NN, weighted by exp(dist)	4.5.1
30	RRBQ		1.0423	0.86737	LM $e_i = 1$, probe+qual as implicit	4.5.2
31	RR2		0.9115	0.86731	LM: combined RRBAS+RRRAS	4.5.2, [Kor08]
32	RRRAS	1-6	0.9361	0.86715	LM using ratings	4.5.2
33	KRRT	7-8	0.8956	0.86710	KRR Gaussian kernel, time-dependent	4.5.3
34	NSVD1R		0.9401	0.86705	NSVD1 using ratings, K=40	4.3.5
35	SVDMT		0.8909	0.86699	SVDM with binned biases	4.3.6
36	log(CMEMB)			0.86695	log(user support)	
37	RSVD2	7-8	0.9064	0.86692	Regularized SVD with biases (RSVD2)	4.3.2, [Pat07]
38	SVDKCOV		0.8844	0.86688	KRR, two time-dep. kernels, K=30+70	4.5.3
39	SRR5	1-6	0.9128	0.86686	RSVD2 postprocessed by approx. RR	4.3.2
40	EMOV			0.86685	Movie MSE of pred. by movie mean	
41	SVDB1			0.86683	Generalized RSVD pred. missing data	4.3.8
42	CMEMB			0.86682	User support	
43	KRR3		0.8846	0.86680	KRR, 4 kernels, time-dep.	4.5.3
44	SVDKRRG	7-8	0.9038	0.86679	KRR, Gaussian kernel $\lambda = 2, C = 1$	4.5.3, [Pat07]
45	SVDKRRG2	7-8	0.9023	0.86675	KRR, Gaussian kernel $\lambda = 0.5, C = 2$	4.5.3, [Pat07]
46	SRR4	1-6	0.9119	0.86674	RSVD2 postprocessed by RR	4.3.2
47	SVDB2A			0.86673	RSVD predicting missing data, ver. 2	4.3.2
48	SVD1N9A	9-10	0.9072	0.86672	RSVD2, reg. sum of movie features	4.3.2
49	RRBAS2	1-6	0.9550	0.86671	Per-movie linear model (LM)	4.5.2, [Pat07]
50	RRBAS	1-6	0.9511	0.86666	Per-movie linear model	4.5.2
51	SVDF7TO12	7-8	0.9971	0.86665	Features 7-12 from RSVD2	
52	NSVD1B		0.9464	0.86664	NSVD1 K=50, $e_i = 1$	4.3.5
53	MOV			0.86664	Movie support rank	
54	KRR3A		0.8846	0.86663	KRR, combined 4 kernels, w/o GE	4.5.3
55	KNN4A			0.86663	Distance to 1-NN RSVD2	4.5.1
56	KME0AVG	1-6	0.9410	0.86663	K-means users	4.6, [Pat07]
57	RRBAL	1-6	0.9505	0.86662	LM with $e_i = 1/\log(\lvert J_i \rvert + 1)$	4.5.2
58	MEMB			0.86662	User support rank	

No	Method	Global Effects	$RMSE_{15}$ individual	$RMSE_{15}$ combined	Description	Described in
59	KME1AVG			0.86662	K-means users on res. of K-means	4.6
60	SVDF1TO6	7-8	0.9325	0.86662	Features 1-6 from RSVD2	4.3.2
61	SVD1N19	7-8	0.9073	0.86662	RSVD K=85, nonlinear postproc.	4.3.2
62	KNN0			0.86662	Residual on GE of 1-NN	4.5.1, [Pat07]
63	RSVD	1-6	0.9155	0.86661	RSVD K=100	4.3.2, [Fun06]
64	SVD4			0.86661	RSVD, modified	
65	DRBM2		0.9175	0.86661	Directed RBM K=100	4.4.1, [Sal07a]
66	SVD1N1	11-13	0.9110	0.86661	RSVD2 K=30, changed no. of iter.	4.3.2
67	SVD5			0.86661	RSVD K=110, modified	
68	NSVD1		0.9328	0.86661	NSVD1 K=40	4.3.5, [Pat07]
69	NSVD2		0.9624	0.86661	NSVD2 K=5 with qualifying data	4.3.5, [Pat07]

All experiments were performed on PC's with 1-2GB RAM and a 1.8-2GHz processor. Most methods needed several hours to train (per method), but a few methods needed up to several days to tune the parameters.

The full blend contained the listed 69 predictors, and additionally 81 two-way multiplicative interactions, which improve $RMSE_{15}$ from 0.86661 to 0.86492. Good practice in regression is including all predictors that form interactions also as individual predictors, but ultimately three variables appeared only in two-way interactions and were not described in the table 41. They are: TIME - the date of the rating, RMOV – the index of the movie in the original, unsorted training set, and NSVDSVD is NSVD1B with $K = 10$, postprocessed by $RSVD2$ with $K = 70$.

While developing subsequent methods, even having best individual accuracy among all implemented methods did not guarantee that adding the method to the ensemble would improve the accuracy. Also, several accurate methods and differing from others, as NPCA [Yu09a] and NREM [Yu09b], did not improve the ensemble accuracy.

Among the above listed $69 + 3$ predictors forming the final ensemble, the following 55 predictors remained from the previous ensemble [Pat07]: RSVD, RSVD2, SVDKRRG, SVDKRRG2, SVDRR, SRR4, SRR5, NSVD1, NSVD2, QNSVD1, NSVD1B, NSVD1R, NSVD2A, NSVDSVD, RRBA, RRBQ, RRBAS, RRBAS2, RRBAL, RRRAS, SVDF1TO6, SVDF7TO12, SVD1N1, SVD1N9A, SVD1N19, SVDB1, SVDB2A, SVD4, SVD5, KNN0, KNN1, KNN20, KNN4A, KME0AVG, KME1AVG, log(CMOV), CMEMB, log(CMEMB), MOV, MEMB, RMOV, EMOV, and all global effects 1-13.

The following 17 predictors were added later: JT2, JT3, KRRS2, KRRT2, KRRT, SVDKCOV, KRR3, KRR3A, SVDM, SVDMT, SVDMT2KNN, DRBM, DRBM2, DRBM3, DRBMKNN, RR2, TIME.

Table 42 lists the used two-way interactions in order of GFFS using the criterion of $RMSE_{15}$ (the same method used earlier for ordering the predictors in table 41).

The two-way interactions were chosen among $69 * (68/2 + 4) = 2622$ possible. This number seems large, but feature selection of new interactions was performed each time after developing a new method, and the set of considered interactions was largely narrowed by the first criterion of feature selection used – statistical significance in the linear regression.

While developing subsequent methods and adding and removing predictors and interactions to the ensemble, different criteria of feature selection were tried (among others, dropterm(), addterm(), stepAIC() in R package, and BIC, AIC criteria). Ultimately, based on experiments with an additional split of the hold-out set, I decided to use two criteria for the feature selection: the first was the mentioned statistical significance in the linear regression, and the second was the amount of drop of the sum of squared error (SSE), calculated on the 15% of probe set, that served as the training set for the linear regression. If SSE dropped less than by a fixed threshold, the predictor (or interaction) was rejected.

Observing table 42 with interactions sorted by greedy feature selection, removing the last 20-30 interactions rather would not reduce the validation error (and perhaps 5-10

predictors from table 41, although they create some of the later added interactions).

Table 42: Two-way multiplicative interactions in the final ensemble, sorted by greedy feature selection

No	Method	$RMSE_{15}$ combined
70	log(CMOV)*KRR3A	0.86635
71	JT2*KRR3A	0.86614
72	KNN1*KNN8	0.86600
73	KNN4A*KNN20	0.86589
74	PME1R*RRBQ	0.86580
75	KNN0*KNN8	0.86572
76	MEMB*RRBAS	0.86568
77	RRBAS*RRRAS	0.86564
78	NSVD1B*NSVD1R	0.86559
79	KNN4A*RRBQ	0.86555
80	QNSVD1*KNN6	0.86552
81	SRR4*RRRAS	0.86550
82	KNN4A*KRR3	0.86547
83	KNN4A*KRR3A	0.86544
84	NSVD1*NSVD1R	0.86542
85	KNN4A*RRBAS	0.86540
86	TIME*RMOV	0.86538
87	NSVD1R*RSVD	0.86536
88	RRBQ*RRBA	0.86535
89	SVD1C*DRBMKNN	0.86533
90	NSVD1R*KNN9	0.86532
91	NSVD1B*KNN8	0.86530
92	QNSVD1*RRBAS	0.86529
93	SVD1D*NSVD1	0.86527
94	SRR4*RRBAS	0.86526
95	PME1R*RSVD2	0.86524
96	KNN4A*SVDM	0.86523
97	SVD1D*RRRAS	0.86522
98	KNN1*exp(KNN4A)	0.86522
99	SRR5*QNSVD1	0.86521
100	EMOV*RRBAS	0.86520
101	EMOV*SVD1N19	0.86519
102	MEMB*RRBA	0.86519
103	NSVD1B*MEMB	0.86517
104	QNSVD1*MEMB	0.86514
105	SVD1C*NSVDSVD2	0.86514
106	SVD1D*NSVDSVD2	0.86512
107	MEANMOVR*NSVDSVD2	0.86509
108	log(CMEMB)*NSVDSVD2	0.86508
109	MEANMOVR*RRRAS	0.86506
110	SVD1C*(DRBM+DRBMKNN)	0.86505
111	RRBA*RRRAS	0.86504
112	NSVD1*DRBMKNN	0.86503
113	KME0AVG*DRBMKNN	0.86502
114	SVDKRRG2*RMOV	0.86501
115	RSVD*RMOV	0.86500
116	CMEMB*NSVDSVD2	0.86500
117	MOV*RRBAS	0.86500
118	KNN1*SVD1NUV1	0.86499
119	KNN0*SVD1NUV1	0.86498
120	SRR4*SVD1NUV1	0.86497
121	SVDKRRG*exp(KNN4A)	0.86497
122	KNN4A*KRRT	0.86496
123	SRR4*SVDKRRG	0.86496
124	QNSVD1*MOV	0.86496
125	NSVD1B*MOV	0.86495
126	SVD5*NSVD1B	0.86495
127	SVD5*KNN4A	0.86494
128	SVD4*KNN4A	0.86494
129	EMOV*SVDKRRG	0.86494
130	PME5R*KNN4A	0.86494
131	SVDRR*exp(KNN4A)	0.86494
132	QNSVD1*CMEMB	0.86493
133	SVD1ND*NSVD1	0.86493
134	SVD1N1*NSVDSVD2	0.86493
135	SRR4*NSVDSVD2	0.86493
136	SVD4*NSVD1B	0.86493
137	log(CMOV)*NSVD1B	0.86493
138	KNN9*KNN6	0.86493
139	KNN4A*KNN1	0.86493
140	PME1R*QNSVD1	0.86493
141	NSVD1*KNN8	0.86493
142	SVD1F1TO6*QNSVD1	0.86492
143	SVD1F7TO12*QNSVD1	0.86492
144	SVD1*QNSVD1	0.86492
145	PME2R*NSVD1B	0.86492
146	SVD4*exp(KNN4A)	0.86492
147	NSVD2*RRBAS	0.86492
148	NSVD2a*RRBAS	0.86492
149	NSVD1B*sqrt(CMEMB)	0.86492
150	SVD5*exp(KNN4A)	0.86492

The linear regression was used for blending without regularization, because, for my ensemble, adding regularization gave no improvement on the validation set. It is possible that if I used regularization (ridge regression), feature selection would not be necessary. As a side note, ridge regression with two-way interactions is similar to kernel ridge regression (KRR) with polynomial kernel, which in turn is close to KRR with Gaussian kernel.

The ensemble listed in tables 40, 41, 42, containing 69 predictors and 81 two-way interactions combined by linear blending, had $RMSE_{15} = 0.86492$. The ensemble was additionally blended in proportion 90%:10% with JT1 model [Tom07] that had $RMSE_{quiz} = 0.8805$ (described in section 4.3.3 as Bayesian PCA). The final validation error after blending was $RMSE_{quiz} = 0.8694$, and $RMSE_{test} = 0.8703$ (8.63% better than the reference algorithm

Netflix Cinematch, with $\text{RMSE}_{test} = 0.9514$), taking 34th place in the Netflix Prize contest, among 5169 competing teams that submitted at least one solution. Without the methods JT1,JT2,JT3 (with a remark that they inspired multiple methods in the ensemble), the validation error of the ensemble was $\text{RMSE}_{quiz} = 0.8707$ and $\text{RMSE}_{test} = 0.8717$ (8.48% better than Cinematch).

Comparing to the best obtained results, the Netflix Prize competition was won by a team of 7 people, Bellkor's Pragmatic Chaos, with an ensemble of more than 450 predictors [Kor09b, Tos09, Pio09, Kor08, Tos08b, Bel07c], that gave $\text{RMSE}_{quiz} = 0.8554$ and $\text{RMSE}_{test} = 0.856704$ (10.06% better than Cinematch). Second place was taken by The Ensemble, a team of over 30 people, with $\text{RMSE}_{quiz} = 0.8553$ and $\text{RMSE}_{test} = 0.856714$ (10.06% better than Cinematch).

Looking at the ensemble of the winners, the main reasons of better accuracy, comparing to our ensemble, seem to be: many more variants of methods implemented, discovery of the frequency effect, more features in the dimensionality reduction methods, extensive automatic parameter tuning, including optimizing the blend accuracy, and better methods of blending. During the contest, our highest reached place were 2nd at the moment of forming the two-person team in September 2007: the method JT1 was merged 50% : 50% with the ensemble [Pat07] enhanced by KRRT. The advantages of our solution over others at that time were that the methods JT1 and KRRT used the date variable, and that the Bayesian model JT1 was exceptionally accurate with $\text{RMSE}_{quiz} = 0.8805$ (7.45% better than the Cinematch).

Summarizing the obtained ordering of features, the first few methods from the table 41 explains most of the explainable variance of ratings. The part of the ensemble containing global effects 1-13 and the first six methods 14-19 has $\text{RMSE}_{15} = 0.86958$, only 0.5% larger than the RMSE of all methods: $\text{RMSE}_{15} = 0.86492$ (the difference in validation error is even smaller, because every feature added to the ensemble causes small overfitting, and decreases accuracy because of that). As discussed earlier, situations of analysis of real-life data can be understood as follows: the data was generated by a certain unknown model, and the optimal method realizes Bayesian inference in this unknown model. We can suppose that the methods found and the identified effects in data, with effort of many people during almost three years of the Netflix contest, provide accuracy close to the best possible. It leads us to the conclusion that it is likely that the optimal method is close to being some combination (not known precisely) of matrix factorization, RBM, kernel methods, and K-NN, all with including the time effects and using the structure of missing data (Conditional RBM, NSVD, SVD++).

To complement, table 43 lists all RMSE_{small} values appearing in the previous chapters. Some methods have too large computational complexity to conveniently learn them on the whole Netflix Prize dataset, or to tune parameters using the whole dataset. In particular, this applies to methods using straightforward missing data imputation. The methods were trained on a small subset of the Netflix data, which is relatively dense. The training set contained exactly $10,000$ users and 500 movies, and 50% values are missing in it. The test set contained the remaining 6% of values – randomly selected $304,500$ ratings. How the small dataset was chosen, was described in section 3.3 "Dataset". The best accuracy among the four methods had KRR with Gaussian kernel, learned by gradient descent. NPCA, which has very good accuracy on the whole Netflix dataset, here gives worse results, probably because the maximum likelihood estimation of the covariance matrix in the EM method overfits the data when averaging over only $10,000$ users.

Summarizing, the contest ended with crossing the artificial limit of 10% improvement of the reference algorithm, many accurate prediction methods were developed, but questions are still open: which single method is the best for the Netflix Prize task? Which probabilistic model generated the data? What is the best possible predictive accuracy?

Table 43: List of experiments on the small dataset.

Method	K	RMSE_{small}	Description	Described in
Global mean		0.9824	Global effect for the rest of the methods	
SVT	80	0.7449	Singular Value Thresholding, adaptive reg., 10 iter.	X.Y, [Cai08]
KRRG	100	0.7415	KRR with Gaussian kernel, gradient descent	X.Y, [Law09, Pat07]
NPCA		0.7520	NPCA without movie bias, 2 iterations	X.Y, [Yu09a]
NNObs	10	0.7482	Heuristic NN between observations	X.Y

The above-listed methods minimized RMSE by modelling the expected rating Er_{ij}. To produce lists of recommendations, I advocate in this work correcting the predicted expected item ratings by multiplicity of their posterior standard deviations $\hat{s}_{ij} = \sqrt{Var\ r_{ij}}$. I have not put much effort into modelling the standard deviation of predictions (I used a non-constant parameter σ for the modelled ratings $r_{ij} \sim N(\mu_{ij}, \sigma^2)$ only in a few models inspired by the JT1 model, which were not included in the ensemble). A rough simplification is estimating by $\hat{s}_{ij} \approx \hat{s} + C/\sqrt{N_j + \lambda}$, and sorting items by $Er_{ij} - C/\sqrt{N_j + \lambda}$ to obtain the top items, with an optional additional correction for the missing data structure. Because sorting items is a situation of multiple comparisons, the constant C should increase with the increasing number of all items. Based on the experiments from section 4.2.4 I roughly estimate, that for the Netflix dataset the best constant should be in range $50 - 500$ (additionally, the constant should be larger for users with few ratings). The resulting personalized ranking of items (personalized recommendations) could be evaluated, but because I proposed only one, heuristic method of calculating recommendations, and because the popular measures evaluating rankings are not completely satisfying (e.g. they ignore the missing data structure), I narrow the experiments to the well defined task of prediction of movie ratings from the data distribution, evaluated by RMSE.

"C'est en faisant n'importe quoi
qu'on devient n'importe qui"

Rémi Gaillard [3]

6 Applications

In the analysis in earlier chapters I paid attention to the perspective of using the outlined prediction methods in a real recommender system. A natural complement of the work will be describing the use of the developed collaborative filtering methods in deployed recommender systems. This chapter describes parts of two applications serving online personalized recommendations. The recommendations are based on the regularized SVD methods, which analysis was the major focus of this work. The recommender system projects, which fragments are described in this chapter, are independent undertakings, and are not parts of this paper's experimental work.

How precise data mining solution is needed, it depends on the application. We can distinguish the following levels of need for recommendations in different applications (see also the discussion of the importance of predictive accuracy in general in section 2.5):

1. No recommendations needed – this is the usual case.
2. Non-personalized recommendations – lists of top items.
3. Any personalized recommendations that fill recommendation slots – can be of average quality, more important is the ease and speed of deployment.
4. Good quality personalized recommendations.

The two applications described in this chapter assumed the fourth level of needs.

The first project is a WWW application, where recommendations are calculated by a server. Recommendations have the form of a list displayed on an HTML website, and are updated with an AJAX-like mechanism. User preferences and recommendations are calculated and displayed instantly after giving a rating by the user, and item features are periodically updated.

The second project is a standalone Flash application containing a set of interactive visualizations, with server-less recommendations, calculated within the application. Recommendations are marked on a 2D map of items. Similarly as in the first application, user preferences and recommendations are updated instantly after giving a rating by the user. Item features, item clustering, and the chosen 2D visualization are precomputed and fixed. The application contains also search by title and filtering by genres, including the non-standard genres defined by the learned SVD features. The set of movies from the Netflix data is extended by a small number of popular movies released after 2006, for which content-based recommendations are provided, based on IMDb keywords. The Flash application, including all data (over 2000 movies and TV series) has a very small size: less than 150Kb.

Observing various emerging implementations of recommender systems by different developers, one can notice common inefficiencies or mistakes, which I strived to avoid here: using K-NN methods instead of more accurate and faster regularized SVD (matrix factorization), using SVD/PCA without regularization, using inaccurate regularization, using SVD for binary data, instead of using generalized SVD, making recommendations by sorting by expected rating without adjusting for variance (uncertainty) of predictions, predicting behavior on training data gathered from a recommender system, without adjustment for missing data structure, relying too much on implicit feedback (passively gathered data), without paying attention to the loopback mechanism that reinforces mistakes made by the

[3] "It's by doing whatever that you become whoever" or "Nonsense makes you become anyone"

recommender system, not ensuring diversity on recommendation lists, not securing against shilling attacks.

The chapter is concluded by discussing the use of collaborative filtering in fields other than recommender systems.

6.1 SVD-based recommender system

The first application is an online SVD-based recommender system in client-server architecture, called "recommender system for everything", recommending web links of any type. The application is an HTTP server implemented in C++/C that serves several websites forming a web interface, and performs additional operations such as calculating recommendations. Similar architecture can be used to realize a recommendation service based on any REST-like protocol.

We can distinguish two ways to serve recommendations: online or offline. As offline recommendations I regard a situation, when the recommendations for a user are updated by recent user votes after some delay. Most often offline recommender systems are implemented by periodically (for example, every day) recalculating recommendations for all users. A disadvantage of offline recommendations is that often a few votes can have visible influence on the inferred user taste, and it would be good to recommend items without delay. It may be especially important for new users (presuming, users yet undecided to use the application, willing to try out the application features) to quickly present personalized recommendations on basis of small gathered information about the user's taste.

As online recommendations I understand computing and presenting accurate recommendations immediately after expressing preferences by the user. The application described in this section is an online SVD-based recommender system, instantly updating user preferences, and periodically updating item features. Similarly as in [Pat07], with item bias $\mathbf{d}$ and item features V held fixed, the user preferences $\mathbf{u}_i$ and user bias c_i are recovered by one-time ridge regression. Recommendations for a user are recalculated, together with the user preferences, each time after a user rates an item, and also periodically for all users, to make use of updated item features. Another possible approach is to update SVD features each time after a user rates an item [Bra03] (quick or instant update of item features may be needed in cases when new items are useful for a user only for a short time).

An alternative collaborative filtering method to SVD are K-nearest neighbor predictions. For the Netflix data, as seen earlier in the work, best SVD-based methods have better predictive accuracy than K-NN methods. Another advantage of SVD-based over KNN-based recommenders is computational complexity and memory usage. In an online system, in the simplest forms of both SVD and K-NN, recalculating recommendations for one user has the computational complexity and the number of memory reads $O(MK + (M - M_i) \log R)$ for SVD and $O((M - M_i)(M_i + \log R))$ for K-NN, where M is the number of all items, M_i is the number of rated items, K is the number of features, R is the number of recommendations. In a typical use the number of features K will be close to the average number of rated items M_i, but the K-NN algorithm will be slower, because memory reads for SVD access always the same fragment of memory, and in K-NN the precomputed distances are read from an area of size $O(M^2)$.

In addition to calculating user preferences and recommendation lists, a recommender system has to initialize the item features, and periodically update the item features. The frequency of updates should be selected depending on the application (type of items recommended), and depending on the number of users, items, and ratings. In a typical online recommender system, 1-2 iterations of updating items per day should be enough, but, for example, in a system recommending news it would be helpful to have new feature values within minutes. Because the recommender system described here had few users, to sim-

plify, I fully re-learned the parameters, using 20 learning iterations, every five minutes. The newly learned item features replaced the old ones after the learning was completed, locking the database only for a very short moment.

To reduce the amount of computation and the memory occupancy without a large loss of quality of recommendations we could take advantage of the usual long tail structure of the dataset of ratings gathered. If there are many items to choose, the set of items used in recommendations can be limited to some number of the most popular ones (because predictions for rarely rated items are burdened with a large variance, those items will rarely enter the list of recommendations anyway), or better, a set of items that high in a non-personalized global ranking. Similarly, with a lot of users, in the phase of calculating item features, users with few ratings can be skipped. Because the calculations of the preferences in SVD are independent tasks when the item features are fixed, and the calculations of the item preferences are independent when the user preferences are fixed, it is easy to parallelize the calculations on multiple processors, for example, using the instruction "parallel for" from the library OpenMP. Also, in some situations, calculating recommendations (or parts of the calculation) can be moved to the client computer – this idea is realized in the server-less application described in the next section. If there are many items to evaluate, another possible speed-up is using "lazy" computation: first calculating approximate personalized predictions for each item, for example, using only the biases, the first (most significant) 5-10 features, and subtracting the variance correction. The approximate scores serve as a filter for the full evaluation – if the approximate score is too low, it is unlikely that the item will enter the user's top-K list, and further calculations for that item can be skipped.

The above outlined concept of an SVD-based recommender system was realized in the mentioned WWW application "recommender system for everything" recommending websites. Users interact with the application through an HTML/JavaScript interface consisting of several pages: logging in, a list of recommended websites, a list of rated websites, and a list of skipped websites. Additionally, an input box was added allowing to add new websites to the system, and a search input box allowing to filter recommendations by keywords from website descriptions (titles).

Figure 34: Screenshot: Recommender system for everything

Figure 34 shows a screenshot – a fragment of a list of rated websites, together with

a website thumbnail sliding on hover over a link. Browsing the lists and the mechanism of rating was realized by an Ajax-like interface, sending XMLHttpRequest queries to the server. The set of items was initialized to several hundred links, including: most frequently visited websites by alexa.com, wikipedia pages, popular movies (IMDb), books, video clips, etc. The application was launched in December 2008 under the names "svdsystem" and "lolrate", was active for several months and had about 300 users.

During operation recommendations were not adjusted for prediction variance (the adjustment would make no visible difference, because the differences between the numbers of item ratings were small). Later, calculating lists of recommendations has been modified by subtracting from the expected rating a multiple of the approximate standard deviation of the prediction. Such adjustment is needed when the gathered item ratings have a "long tail" structure, as explained in sections 4.2.4 and 3.4.

On the implementation side, "recommender system for everything" is a Linux-based HTTP server, implemented in C++/C with occasional use of Fortran libraries. The application serves several HTML and XML pages, updates the database, and calculates recommendations. To allow rapid access to the stored data, the database is realized as a file mapped to memory with the function mmap(), treated as shared memory with the access guarded by a mutex lock. The server is process-based, similarly as the Apache server, but without a speed optimization by keeping a thread pool. Other realizations are possible, such as thread-based or one-thread with epoll() listening, but in these two variants an obstacle for a high-performance server would be a limit on the number of open files in Linux.

The algorithm used was a variant of the basic regularized SVD with constant-linear regularization. The application contained a procedure performing automatic tuning of the regularization parameters, intended to be periodically run. An obvious shortcoming of the regularized SVD is not taking into account the structure missing data. Importance of missing data is visible, for example, when calculating the average rating of movies (see discussion in section 4.2.4). A partial fix is using methods like SVD++, which modify user preferences by a prior depending on missing data. Another, complementary solution could be a heuristic used in the recommender system described in section 6.2.3, penalizing unknown (rarely rated) regions of the set of all items. It remains to be determined, how important is the correction for missing data to the ultimate goal of producing an ordering of items useful and satisfying for the user.

If a recommender system starts from a small amount of data, this situation is called "cold-start problem" (mentioned in section 3.1). In such a situation it is particularly important to make best use of available data, including the additional information about items and users. In the described "recommender system for everything" to support overcoming the cold-start problem well worked a heuristic solution of creating artificial users, who "like" an artificial subset of items, e.g. action movies, and "do not like" a random subset of remaining items. For larger groups of items (such as movies or books) 3-4 artificial users were added who "like" all items in the given category, and for smaller groups of items one artificial user was added per group. This heuristic method has the advantage that it does not require creating a specialized model using side information about items – a purely collaborative filtering recommender system can be used unchanged. Adding artificial users can be performed by a person untrained in machine learning. The heuristic of artificial users worked satisfactorily well, but useful would be evaluating it in comparison with model-based use of side information (more about using metadata in section 4.7).

In services gathering additional user data (the described recommender service did not) an analogous technique of creating artificial items can be used, to help with the cold start problem for new users. The idea of adding artificial users or items is similar to using conjugate priors (which behave as additional observations) in probabilistic models.

The above-described recommender system did not provide diversified recommendations (does not avoid recommending very similar items). A simple idea to correct for diversity is to use clustering (the possibility mentioned in [Pat07]), for example, to limit the recommendation list to up to one item from one cluster. Another possibility is recommending whole clusters (this option was used in the Flash application described in the next section). Instead of using precalculated clustering, we could remove one-by-one items that are very similar to other items higher on the recommendation list.

Another not implemented feature, but that could be useful to adapt the described recommender system to different domains, is detecting identical or equivalent items and removing them from recommendation lists, or merging them in the database. Many other appearing issues to tackle in a recommender system were listed in section 3.1.

One can wonder what is the best number of features to learn in the regularized SVD. Based on own observations about what information the features learn, and what is the impact of individual features my suggestion is: 16-32 features should be enough to sufficiently explain user preferences in most single-domain applications as movies, music, books, but more features, up to 100-200, are needed in broader, cross-domain recommender systems, as the described "recommender system for everything". Because SVD is not capable of fully modelling a large amount of local item-item correlations, it may be helpful to augment the factorization model with a neighborhood-based component.

6.2 Using distance between items

Matrix factorization methods give a reduced dimensionality representation of each item in the form of a K-dimensional vector. A well performing measure of similarity between items is the Euclidean distance between vectors, or between normalized vectors (called also the cosine similarity). Those similarities can be calculated quickly, and this gives us possibilities of various useful applications.

In this section I will describe a Flash application containing interactive visualizations of distances between items, and recommendations on a 2D map, besides a traditional list of recommendations. The parts of the application will be described: the clustering of items, the visualization of items on a plane, and the "2D recommender system". Recommendations are computed inside the application, instead of being calculated by a specialized server.

6.2.1 Clustering

Clustering, that is grouping similar objects together, has some applications in recommender systems, for example, clustering can be used to improve the user interface, clustering can help to add diversity to recommendations, as we mentioned, or to speed up the collaborative filtering algorithm at the cost of accuracy.

The 2D recommender system described here uses clustering for two purposes. One is to reduce the number of displayed points on the 2D map, displaying clusters instead of individual items, and giving in effect a more convenient and faster interface, and reducing the overwhelming choice for the user. The second purpose is that whole clusters are recommended. The 2D recommender system recommends 10 clusters of movies at a time, instead of 10 movies. A list of best clusters is more diverse than a list of best movies, and personalized predictions within one cluster are similar anyway.

How to create good quality clustering for the Netflix data, and what "good quality" clustering means? Like other tasks of data exploration, the answer depends on the application, and is partially based on experience. One could try to formulate a criterion evaluating a clustering, that rewards situations when items in one cluster are rated by users similarly, and rewards clusters of certain size – penalizing too big and too small clus-

ters, but instead of optimizing a chosen cost criterion, I tried out several commonly used clustering algorithms on the K-dimensional space of item features coming from the learned regularized SVD (the model used for recommendations). In [Pat07] it was mentioned that good results are obtained with single linkage hierarchical clustering using the Euclidean distance between item features. In later experiments it turned out that a more balanced clustering is obtained using the k-means method, initialized to a clustering calculated by a greedy heuristic. The resulting clustering was used in the sub-applications described in the next two subsections (interactive visualization and 2D recommender system). The clustering by k-means can be further improved by limiting the maximal size of a cluster, for example, by repeatedly, randomly splitting the largest cluster in the last iterations of k-means.

6.2.2 2D visualization

The described set of Flash tools contained an embedding of movie clusters on a plane, used in interactive visualizations to explore similarities between movies, and used in the associated "2D recommender system".

Visualizations on a plane are used very often in data exploration. Visualizations are addressed to a human, and their purpose and to communicate information in an efficient way, and to emphasise important aspects of data. Locations of points, colors of points, size, and shape seem to be a more natural medium of communication (their reception requires less effort) than using words, numbers, lists.

Here we focus on visualization of data in two dimensions by using simple mediums: points and lines, with varying location, color and size. A properly chosen visualization allows to present the data in a way that allows to see interesting properties of the data, where what "interesting" means needs to be clarified – it depends on the application, depends on what we expect to find in a given kind of data. The perspective of a user unaccustomed with mathematics and machine learning, who just wants to explore similar movies, or get personalized recommendations, differs from the perspective of a data analyst, who wants to utilize visualizations to improve a developed data mining solution, gain deeper insights into the data. Both perspectives were important while developing the tool. As for the data analyst perspective, the intention here is to find ways to further improve the regularized SVD models, find new effects, reduce the number dimensions from the initial 32, discover a meaningful clustering of items, identify the right nonlinear transformations of features, identify probability distributions in data. Realizing exploration and model identification goals, such as the mentioned above, can be difficult when working only with numerical data. Usually it is more effective and convenient to utilize the pattern matching abilities of the human brain, and use visualizations. Various kinds of visualizations, from histograms, boxplots, quantile-quantile plots, to scatter plots, spline smoothing, regression diagnostics, were extensively used while developing the ensemble described earlier in the work.

The visualization developed here compresses the 32-dimensional vectors of movie features, coming from the regularized SVD, into two dimensions. The 32-dimensional feature representation from SVD happens to define a well working distance between items, by taking the cosine similarity between feature vectors [NF07, Pat07]. Differing forms of the regularized SVD model with different learning methods give much the same distances between movies. I decided on using a Variational Bayesian version of SVD, with fully learning one feature at a time (requiring 10-20 iterations for each feature), and repeating the learning of all features three times. The property, that the features are roughly sorted by the variance explained, was useful in the second implemented heuristic visualization (not described here) presenting a separate map for each cluster, centered on that cluster. Both visualizations have been designed to make it easy to spot the nearest movies (I presumed that the user is interested primarily in searching for similar movies).

Besides visualization, distances can also be used directly to make predictions, as in the described earlier (section 4.5) K-NN methods and kernel methods.

The first of the interactive visualizations (the description of the second one will be skipped) is a map of 2000 movies grouped in 487 clusters. Clusters are displayed as circles with the diameter proportional to the cubic root of the cluster size. The visualization is interactive – after selecting a movie in the Flash application, all clusters are colored according to the distance from the chosen cluster, and additionally lines are drawn to four nearest clusters. As the movie-movie similarity (distance), the cosine similarity between cluster features was used: $s_{ij} = \frac{\mathbf{v}_i^T \mathbf{v}_j}{||\mathbf{v}_i||_2 ||\mathbf{v}_j||_2}$. The cluster features $\mathbf{v}_i$ are averages of the normalized features of all movies included in the cluster. I stored their normalized versions in the application: $x_i = \frac{\mathbf{v}_i}{||\mathbf{v}_i||_2}$

The visualization task, which we set ourselves, is to choose one of the possible ways of representing 32-dimensional vectors in two dimensions, in a way that the objects close in the 32-dimensional were also close on the plane. Such types of visualization tasks are often formulated as force-driven algorithms (calles also force-directed, force-based), that is, forces acting on each point are chosen, and the resulting dynamical system is simulated, for example, in discrete time increments. The item-item distances in the 32-dimensional SVD space can be changed to forces, for example, according to Hooke's law, as if the items were connected with springs, and too close distances on a plane can be penalized, for example, by using forces modelled after electrical repulsion by Coulomb' law. Force-directed graph drawing was used to visualize the Netflix data in [Ste10a], where the graph was defined by similarities calculated directly from ratings, and in the visualization [Gan09], where the graph was defined by 10 nearest neighbors according to the matrix factorization similarity.

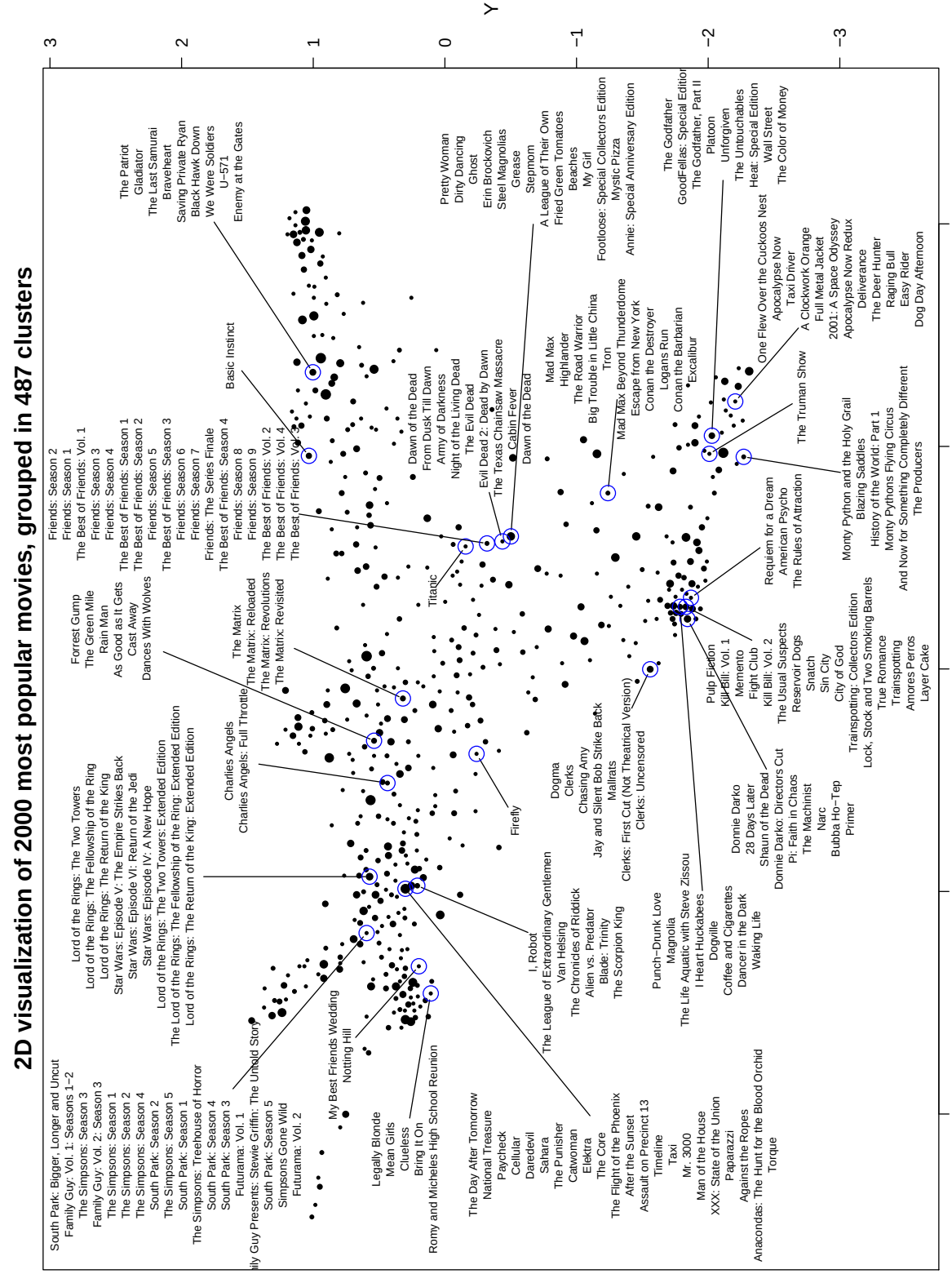
Commonly used alternatives to force-driven methods are multidimensional scaling methods, and spectral embeddings – see visualizations of similarities between music artists in [Gle06] or visualizations of graphs (networks) in other domains [New03].

I used a simpler, heuristic iterative approach, where each iteration consisted of three phases: In the first phase, for each subsequent cluster, its coordinates are updated to the weighted average of four closest points according to the distance s_{ij}, with weights $\overline{d}_{ij} = \exp(10 s_{ij})$, where s_{ij} is the cosine similarity between the feature vectors of clusters i and j. The second phase is the repulsion between points, where the vector of change is $v_{ij} = \hat{d}_{ij}^{-3}$, where $\hat{d}_{ij}$ is the Euclidean distance between clusters i and j on the plane. In the third phase, the coordinates are scaled and shifted, so that they will fit within a specified part of the plane. The resulting visualization, selected from one of several runs of the algorithm, is shown on the plot 35.

The visualization balances between the requirements: items similar according to s_{ij} should be close in the visualization, but items should not be too close to each other, and the resulting set of points should be diverse, not too monotonous (visualization is more appealing, and also more useful, when it consists of varied, easy to distinguish shapes, rather than when it consists of repetitive shapes).

A downside of the chosen method of visualization is that unrelated clusters are often close to each other, for example, the cluster including "Dawn of the Dead" is close to "Friends" and "Pretty Woman". This can be partially fixed by adding a penalty to the cost function for closeness of non-similar clusters, and the obtained more complex cost function could be optimized with simulated annealing, which has the capability of escaping local minima. Another way to avoid local minima is to initialize the cluster positions with a result of another algorithm that would properly deploy large groups of similar clusters (for example, we could create 10-20 groups of clusters using an additional run of the k-means algorithm).

Figure 35: 2D visualization of movie clusters.

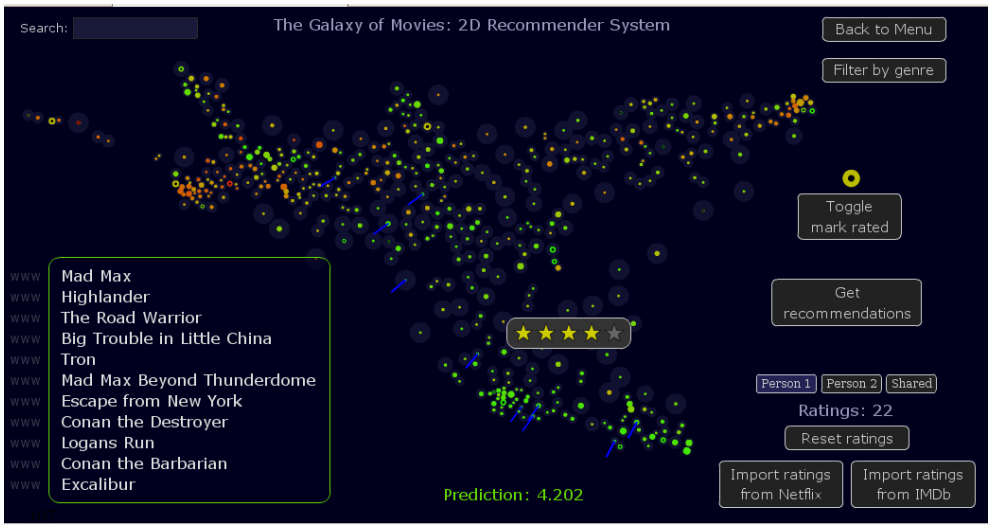


6.2.3 2D recommender system

We can look for alternative ways to present recommendations than displaying a simple list. One possibility is presenting recommendations by displaying points or marking areas on a plane. The intention is that it can be easier for a user to remember a point on a map, than to traverse a complicated structure of catalogues, lists, tags, or even to use a text search. In a good 2D visualization points close on a map should roughly correspond to similar movies. It can be expected, that the user ratings in groups of close movies will be similar, making it easy for a user to notice regions of the 2D map, where lie the movies liked by him. The advantage for recommendations is also the potential capacity of 2D visualizations, the possibility of presenting a lot of items on a small space. For example, the Flash application described here displays simultaneously 487 clusters.

In the Flash sub-application "2D recommender system", the visualization of clusters (described in the last section 6.2.2) was augmented by recommendations. Recommendations are presented in two ways: coloring all points-clusters according to the predicted rating, and, after clicking a button, explicitly marking on the map the clusters from the list of top recommendations, in groups of 10 clusters. Picture 36 shows a screenshot of the "2D recommender system".

Figure 36: Screenshot: 2D recommender system



The recommendations are server-less, that is, user preferences and recommendations are calculated inside the application, without queries to a specialized server. The recommender system is SVD-based: the regularized SVD model is trained once with Variational Bayes (see section 4.3.3), one feature at a time, and the resulting features are stored in the Flash application, one normalized averaged vector per cluster (see the previous section 6.2.1 "Clustering"). Inside the flash application, user preferences are calculated with a regularized SVD with constant-linear regularization, learning one feature at a time by a jump to the marginal minimum of the cost function (the method described in section 4.3.2). To produce recommendations, items are sorted according to the formula: $c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j - 10\hat{s}_j - C_{ij}$. The parameter $\hat{s}_j$ is the estimated standard deviation of predictions for the cluster j (or rather the difference between the standard deviations in different clusters), crudely approximated as $\sqrt{\alpha_j}$, where α_j is the average of the values $(N_{j_2} + 10)^{-1}$ in cluster j, N_{j_2} being the number of ratings given to a movie j_2. The C_{ij} term is a heuristic correction for the missing data structure, giving more penalty to the prediction of ratings that are more likely to be missing: $C_{ij} = 0.7 - max(0, \rho_{ij})$, where ρ_{ij}

is the similarity (cosine similarity between two vectors of features) to the nearest cluster j rated 4 or 5 by user i. User preferences and recommendations are recalculated and shown instantly at the moment when the user rates an item. Ratings given by the user do not influence the item features, that means, item features are fixed all the time.

Other corrections for the missing data structure are possible. An untested heuristic correction is using Mahalanobis distance to the set of vectors v_{j_2} rated 4 or 5 by the user: $C_{ij} = \alpha\sqrt{(\mathbf{v}_j - \mu_i)^T S_i^{-1}(\mathbf{v}_j - \mu_i)}$, where μ_i, S_i are the parameters of the estimated multidimensional Gaussian distribution approximating the distribution of points v_{j_2}. The constant α can be hand-picked to produce best recommendations, as evaluated empirically. Instead of using the C_{ij} adjustment term, another way to correct the method may be adding a small number of low ratings for each user, for items drawn uniformly at random [Ste10b] (a disadvantage of this method is that it penalizes unpopular items), or for items drawn according to their distribution in the data [Dro11], still leaving the risk of adapting too much to the structure of missing data, which may be not entirely related to the user preferences. Ideally, recommendations should be tested on random items that users have to rate or indicate that they do not know them (the Netflix dataset does not contain such data).

The accuracy of recommendations in the application could be further improved to a small extent by predicting user preferences using the missing data structure (see the algorithm SVD++, section 4.3.5) and by using the time information (section 4.3.6). Because movies are grouped into clusters, and the recommendations are for whole clusters, the algorithm could be improved by incorporating the clustering within the model, and learning the features directly for the whole clusters, instead of postprocessing the regularized SVD results and averaging the SVD features for groups of movies.

Observing visualizations of predicted ratings in the application I noticed that sometimes giving an extreme rating (low or high) does not influence enough the predictions for similar items, even when no ratings are given for similar items. It indicates that the used regularized SVD algorithm, and in particular the one-dimensional ridge regression used to infer a user preference for a feature, may be too robust to outliers. A remedy could be optimizing a different loss function than MSE, with a higher penalty for large errors, for example $|\hat{r}_{ij} - r_{ij}|^p$ with $p = 3$ or $p = 4$, but I have not tested this idea.

Several additional features were implemented in the application: search by title, filtering by genres, including new genres defined by the six largest SVD features (see section 4.3.4), recommendations shared for two persons, calculated with a heuristic that combines two sets of ratings, and the option of importing ratings from the services Netflix and IMDb.

A similar idea of visualizing recommendations on a map for 1000 most popular TV shows was described in [Gan09]: the map is generated dynamically based on user preferences (inferred from the data about watched TV shows), and based on which TV shows are available. The visualization in [Gan09] was based on force-directed graph drawing of a network of 10 closest items according to similarities taken from a factorization model built on users' preferences for TV shows. In [Gan09] the following ways of recommendation were proposed: a user-driven way, where a user can explore a heat map, coloured based on the user's preferences, and a user-passive way, where recommendations proposed by the system are marked on a map, together with an explanation by indicating the watched similar TV shows. Comparing the Flash application described here to the approach in [Gan09], they differ in: the method of visualization (here movies are clustered by k-means, clusters are displayed as single points, and the 2D map is heuristically generated), in the method of generating and visualizing recommendations (server-less instant recommendations, coloring points according to user preferences, explicit recommendations by marking ten points-clusters), and in the user interface and available features.

6.3 Beyond recommendations

The task of predicting missing values in a sparse matrix appears also in domains other than recommender systems, and it is justified to expect that for some of those datasets and tasks effective will be collaborative filtering algorithms similar to the analyzed in this work. Particularly interesting is the possibility of using dimensionality reduction algorithms. Examples of successful applications include using collaborative filtering for medical data [Has10], educational data [Tos10], image processing, or text analysis.

I implemented collaborative filtering algorithms for two datasets outside of the field of recommender systems: for outcomes of soccer matches, and for outcomes of chess games. Both attempts did not give better prediction than the known models, but the experiments are worth mentioning, because such negative results with overly flexible models to some extent support hypotheses, that the best known models are close to the optimal.

The first dataset was soccer matches from years 1994-2009 from English Premier Division and English Divisions I and II. As a baseline model was used the proposed in [Hav97] model $H_i \sim Poisson(b + a_i - d_j)$ and $A_j \sim Poisson(a_j - d_i)$, where H_i are the goals of the team i playing home, and A_j are the goals of the team j playing away. The parameter b models the home field advantage. The parameters a_i can be understood as the attack skill of team i and the parameters d_i as the defense skill of team i. The attack and defense parameters change in time, and the change is well modelled by exponential smoothing. In my implementation the parameters were learned by alternating point estimation using exponentially smoothed, regularized Poisson regression. The resulting accuracy was slightly worse than the accuracy of bookmakers' odds (included in the dataset), with a remark that the implemented model used only the variable of goals, without using any additional variables, such as shots on goal.

The tried out collaborative filtering model for soccer matches had the form: $H_i \sim Poisson(b + \mathbf{u}_i^T \mathbf{v}_j)$, $A_j \sim Poisson(\mathbf{u}_j^T \mathbf{v}_i)$, learned by alternating regularized Poisson regression, with exponential smoothing in time. A simplification made here was using a constant regularization parameter (as the experiments with the Netflix Prize show, when hidden variables are combined by multiplication, it is better to use constant-linear regularization, or an approximate Bayesian approach, such as Variational Bayes or MCMC; also, more accurate can be treating the hidden *lambda* parameter as a random variable with Gaussian noise). In the experiments, using more than two features did not improve accuracy, and the two-feature model had about the same accuracy as the baseline model of two biases. It is an argument supporting the hypothesis that the model of two biases is close to the unknown optimal model, and no additional multiplive components are needed (an interpretation is that soccer teams cannot be divided into groups with largely different way of play, as we can do with movies by distinguishing movie genres). Of course when modelling real-life data, we can never entirely exclude the possibility that we overlooked some important pattern in the data.

The second modelled dataset was the Kaggle dataset of 65,053 chess games by 8,631 players, selected among world's highest rated 13,000 chess players. Several algorithms can be regarded as baseline algorithms for this task: ELO, Glicko [Gli99], Chessmetrics or TrueSkill [Her07]. The model I have tried out had the form: $p(Y_{ij} = 1) = 1/(1 + \exp(b + c_i - c_j + u_i v_j - u_j v_i))$, where Y_{ij} is the game result, b is a global parameter accounting for white player's advantage, c_i bias variable corresponding to the one-dimensional playing strength of player i, and u_i, v_i create the "collaborative filtering" term, added in order to see if an additional dimension of playing strength helps. The parameters were learned with alternating regularized logistic regression with a special prior depending on the number of matches played, depending also on a weighted average of opponents' ratings, and exponentially smoothed in time. It turned out, that the feature u_i, v_i does not improve the accuracy of the model. It is an argument, that the right approach is using only one variable

(one ranking) to model the playing strength of a player. The used data were released in a prediction contest organized by Kaggle, which was followed by another contest with 1.84 million games of 54,000 chess players. The best models in both contests contained one ranking variable per player (though with additional parameters and effects).

In [Sta11] models with 1-15 factors were used to predict results of games of go, with non-conclusive results, whether increasing the number of factors improves prediction.

A question remains open, whether in any popular sport collaborative filtering models can be useful to model the playing strength (perhaps in poker?). In such sports cyclic relations should occur, such as: the team A is better than the team B, the team B is better than the team C, and C is better than A. If we do not observe such situations, it is likely that the optimal modelling of results will be limited to models of biases, as in chess (one main variable per player – the player's skill) or in soccer (two main variables per team – the attack skill and the defense skill).

Interesting would to check if models with dimensionality reduction are useful for modelling stock market data or currency data.

"Science and technology provide the most important examples of surrogate activities"

Theodore Kaczynski

7 Summary

I discussed, in the context of recommender systems, the most accurate collaborative filtering methods predicting ratings, I summarized own experiments, and experiments of others on the Netflix Prize dataset. The focus of the experiments was on obtaining possibly best accuracy in the associated prediction task of minimizing RMSE. The large size of the dataset (100M ratings) makes it possible to compare accuracy of a large number of methods, allows to use methods with more parameters, and gives a larger capability to identify the underlying probabilistic structure that generated the data, than it is possible for typical publicly available datasets, most being of size 100-10000 observations. Since making the Netflix Prize dataset available, many people worked independently on the same prediction task (the contest website states that over 5000 teams submitted their solutions), and as the result the Netflix Prize task has been exceptionally well analysed relatively to other prediction tasks on other datasets. A thorough analysis of one prediction task gave methods and insights that are useful not only for collaborative filtering recommender systems, but should generalize well on other prediction tasks in other domains. No need to explain that the need for accurate and simple methods, and time-efficient approaches for data analysts appears in a huge number of applications, and it would be good if large needs were matched with an in-depth understanding of the field of prediction.

The task proposed by Netflix was to minimize RMSE between predicted ratings and the real observed ratings on a hold-out set. The task was a compromise between simplicity and being relevant for developing recommender systems. The choice of RMSE as the evaluation measure caused that our task of analysing a real-life dataset was relatively "clean", while being a good intermediate step in the application to serve lists of recommendations to users. It turned out that there is depth in the simply formulated task of predicting ratings, and the challenge we set ourselves, of developing possibly most accurate methods, contains certain necessary complexity, which I attempted to explain in this work.

Summarizing shortly the state of understanding of the Netflix task, the most accurate methods combined modelling the structure of the data on three scales – dividing according to the number of parameters: the global scale, including global effects such as biases, the middle-level scale with dimensionality reduction structure (here the most accurate models were SVD, RBM and KRR), and the most parameterized scale explaining direct item-item relations, by neighborhood models, and also by kernel methods not using dimensionality reduction, such as NPCA. In the most accurate methods, each of the three scales included time-dependent variables, which captured the model variability over time. In addition, a significant improvement of accuracy was obtained by using the structure of missing data, that allowed to some degree to predict the user preferences before seeing the user ratings.

In the work I studied closer the middle-level scale containing the most visible development coming from the Netflix contest – the structures with dimensionality reduction, on which the most accurate methods predicting ratings were based. In particular, the developed matrix factorization methods are typically more accurate and faster than the earlier applied [Mon03] near-neighbor-based collaborative filtering recommender systems, and already resulted in numerous applications. It should be noted that datasets in different domains do not always contain a structure with reduced dimensionality, but it seems that in datasets with gathered users' preferences for items such structure with low number of dimensions (but more than just global effects) usually exists, intuitively understood as a

division of items into groups that a user can like or dislike, for example, the division of movies into genres.

Among the most accurate models (MF/SVD, RBM, KRR, NPCA), the ones most commonly used were based on matrix factorization methods, that is the models containing sums of multiplied variables: a continuous variable representing an item feature (an automatically learned movie genre) multiplied by a continuous variable expressing a user preference for that feature. In this study, in particular, I examined closer the variants of MF that learn one variable at a time, called here regularized SVD. Two basic ways of learning the parameters were described: neural networks approach with special regularization, and the approximate Bayesian approach, as Variational Bayes or MCMC. An observation important for the whole domain of neural networks is that multiplying hidden parameters necessitates the use of another regularization than the standard, constant weight decay – necessary is the amount of regularization growing linearly with the number of observations [Fun06, Lim07, Rai07].

It can be disputed, whether the usual probabilistic matrix factorization with Gaussian priors is the proper hidden structure to model movie ratings (or to model any other data). For example, the VB approximation gives posterior Gaussian distributions, meaning that a rating is approximated by a sum of products of two Gaussian variables, which seems to be a quite unnatural choice. Experiments with nonparametric priors were performed, which dispute the choice of Gaussian prior for the hidden parameters of the matrix factorization. Also, experiments were done with learning a nonparametric relationship predicting the rating, which dispute the choice of multiplication as the operation connecting the hidden parameters (this is redundant with changing the priors). The experiments confirmed inaccuracies of the probabilistic model of matrix factorization, but have not led to changes that visibly improve the accuracy.

We can say about each real-life dataset, that it was generated by a certain unknown model. Data analysis tasks, in particular prediction tasks, rely heavily on the identification of the unknown model, that means on discovering the right structure of parameters, dependencies, patterns, probability distributions, effects in data – identifying all elements relevant for the purpose of the analysis (in our case, obtaining the best prediction accuracy). An approach fully Bayesian would be to impose a prior distribution, specifying the probability of every possible model, and to apply Bayes' rule to infer the posterior probability distribution over models. Such two-step attempts with defining all data analyst's prior beliefs are impractical (even assuming, that we could perform the Bayesian inference exactly, without computational difficulties). In practice, experience shows that the right approach is to iteratively refine the model, based on how well it fits the data (or based on the predictive accuracy), hence the data is used multiple times to exclude implausible models, and we concentrate the effort on exploring plausible models, supported by the data. We can say that we need to optimize, on the meta-level, the amount of analyst's time and effort put, needed to identify well enough the model that generated the data, or to find a method accurate enough for the given purpose (the decision criterion in prediction can be optimized directly, without the intermediate step of building a generative or discriminative probabilistic model). The experience from the Netflix contest was that a convenient framework to search the set of possible models was to blend many models (in the Netflix task blending with linear regression or ridge regression worked well), which gives a combination of methods with accuracy much better than the accuracy of individual models. Blending allows to assess contribution of each method, remove unnecessary methods, and focus on those that improve accuracy of the ensemble. When a method improves the blend, it suggests us that there exists a way to integrate this method with one or more remaining methods, creating another method with better accuracy. The best accuracy in the Netflix Prize task was reached by ensembles containing hundreds of models.

In the process of searching for the best models useful was automation, mostly for automatic parameter tuning by minimizing the hold-out test set accuracy (I used for tuning the Praxis procedure from the Netlib library). A well performing technique, unused in this work, was choosing the parameters to optimize the blend accuracy, instead of optimizing the accuracy of individual methods [Tos09, Pio09]. An open issue is how to automatically search (test) large classes of models, (for example, a class of multitask models with one layer of hidden variables, like models MF, RBM, PLSA, and similar), or to test a large set of possible effects. The larger the dataset, the greater are the possibilities of automation, and for the dataset of the size of Netflix's, automatic testing of plausible classes of models and effects should be to some degree possible to realize. This is a topic for future research. Is fully automated model identification possible? I am skeptical about it – in real-life tasks domain knowledge always will be needed to limit the range of tested models. Interesting was absence of methods based on decision trees in ensembles for the Netflix Prize (except for the use for blending predictions). Decision trees seem to work better as a black-box method for small datasets with lots of predefined features, when a quick analysis is needed, without precisely identifying the underlying model.

We can hypothesize about, to which degree (thanks to the effort of many people) the optimal model has been discovered for the Netflix dataset and task. Looking at the best combinations of methods in the greedy feature selection in the table 41 it can be supposed, that, if some important method did not remain undiscovered, the unknown optimal model, about which we can say that it generated the data (or rather the part of data $p(rating|user, movie, time)$, which we want to predict), is some combination of SVD, RBM, KRR, and neighborhood models, with global effects, improved user preferences, and effects explaining the model variability in time.

In most situations of estimation or inference encountered in the Netflix Prize task, as well as in other prediction tasks, necessary is an appropriate regularization. The methods of the classical statistics, such as point estimation with uniform priors (the maximum likelihood method), are inexact, result in a non-optimal prediction. What's more, the experiments with approximate Bayesian approaches, as MCMC and VB, show that the whole idea of point estimation of posterior distributions is sometimes largely inaccurate, and necessary are corrections, as, in the models with hidden variables for the Netflix Prize, we needed to use an amount of regularization growing linearly with the number of observations.

An advantage of the choice of RMSE as the evaluation measure was the simplicity of the resulting algorithms optimizing it, for example, efficient was blending methods with the linear regression, or modelling ratings by sums of biases, linear, and bilinear terms. Having a fixed criterion of accuracy allowed to compare results objectively, which certainly helped in that the effort put by many people resulted in analysing the task in depth. As argued in the work, predicting ratings by minimizing RMSE is a good intermediate step to a useful real-life goal, calculating good quality lists of personalized recommendations. Recommender systems differ, and multiple issues may be important as the goal of recommendations: the type of data collected, diversity on a recommendation list, diversity in time, balancing popularity vs. novelty, dealing with cold-start situations, resistance to shilling attacks, integration with with search, categorization, tags, social navigation, explaining the recommendations. Looking at the above vast list, it is evident that many aspects of calculating recommendations have to be simplified. The crux of a recommendation algorithm is always to guess in some way the preferences of a user for items, thus (although not always directly) to predict ratings accurately, but it is not enough to accurately estimate the expected rating. Calculating relevant recommendations is rather a combination of several machine learning tasks, such as estimating, besides the expected rating, the uncertainty of predictions, the probability that a rating is missing, the prob-

ability that a user watched the movie, the influence of the missing data structure on the expected rating and on the uncertainty of predictions, predicting the user's short-term intent using entered search queries, clicked tags, currently visited item page, etc., estimating similarity between items or users, detecting duplicate items, matching items in different databases. To generate lists of recommendations in an easy way using algorithms such as the developed in this work, a well performing method was to sort items by specially adjusted expected ratings. If the items in a recommender system have very different support (this is usually the case), it is necessary to adjust the expected rating by the estimated standard deviation of predictions. The adjustment is the more important for users with few ratings, and results in recommending more popular content for those users. If recommendations are chosen from a large set of items (more than thousands of items), another adjustment depending on the probability of missing data may be needed, because the algorithms minimizng RMSE on the training set distribution give too large rating estimates for ratings which are likely missing (usually this bias affects items with low predicted rating, but because of uncertainty of the predicted rating, the more items are in the system, the more often the items with heightened prediction will enter the recommendation lists – a situation which we want to avoid; additionally, the uncertainty of predictions made is higher for the data that is likely missing). A heuristic solution proposed for the adjustment for the missing data bias was to appropriately define and use a distance to the set of items highly rated by the user. More research and experience is needed to determine the appropriate criterion of evaluation and comparison of personalized rankings (lists of recommendations), and to determine if it is worth to optimize such ranking-based criterion directly. Accurate algorithms optimizing a proper ranking-based criterion probably would be similar to the developed most accurate methods for the Netflix task (see, for example, [Jah11b]). To verify it, more experiments are needed on specially gathered new data.

The recommendation method advocated in this work is two-step. First, the regularized SVD model $c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j$ is learned to predict ratings from the observed data distribution. I preferred to learn the parameters with the Variational Bayesian approach or with its neural-networks-like simplification with a "constant-linear" regularization term, and I preferred to fully learn one feature at a time (with learning all features repeated 2-3 times) to concentrate the learned variability of ratings in the first features. Informative priors can be used for the user preference parameters u_{ik} (for example, based on the implicit feedback or on demographics), and for the item features v_{jk} (based on the movie metadata – improves RMSE accuracy only for items with few ratings). In the second step, having the model predicting ratings, recommendations for user i are calculated by sorting the considered items j by the following score (partial sorting to select the top items is used, of all items, or of a context-based subset, filtered by search, selected category or tags):

$$c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j - \alpha s_j - C_{ij}$$

The adjustment αs_j penalizes the uncertainty of the prediction. The adjustment C_{ij} corrects predictions for the missing data structure, penalizing those items that are unlikely to be known and rated by the user. Heuristics for s_j and C_{ij} were proposed in sections 6.1 and 6.2.3. The obtained list of top items can be postprocessed to ensure more diverse recommendations, if needed.

I will end with a recommendation how to best divide efforts when faced with similar tasks of real-life data analysis, to make best use of available data:

- Choosing the right task, 60% efforts – researching what is the proper task, what is the right problem to be solved, how to collect data (various observational studies or experimental designs are possible), discover good candidate variables, how to evaluate results. To concretize the task helpful are: good domain understanding, observation of users' interactions with the developed tool or service, conducting

surveys among users. In the task chosen to analyze in this work, Netflix has done a good job with gathering the right kind of data – ratings, which allow a user to express positive and negative preferences. Ratings are a better type of data, closer to what user thinks, than passively gathered information, such as clickstream data, or links between pages, which are the basis of commonly used search algorithms. Netflix also proposed a good evaluation criterion, relevant to the real-life goal of preparing good quality lists of personalized recommendations. Of course, measuring accuracy by hold-out test error is one of the easiest ways to formulate a prediction task. In practice, it is important to consider how the situation where the data is gathered is related to the situation where the developed solution is used. When calculating lists of recommendations, items are evaluated within a different missing data structure than the structure of training data. To what extent the algorithms optimizing hold-out RMSE have to be corrected for the missing data structure to produce recommendations, this is a subject for further research.

- Model identification, 30% efforts – understanding the data, guessing the underlying probabilistic structure, discovering meaningful effects, patterns, dependencies in the data, discovering to a sufficient degree the model that could plausibly generate the data. For example, guessing the variable distribution is a non-obvious task for of a one-dimensional, directly observed variable. All the more difficult is guessing the distribution of hidden variables, where, no matter how much data we gather, there is always substantial uncertainty left, whether we model the hidden structure properly. We can use techniques such as starting from a nonparametric or overparameterized model, and then, based on the learned shape, to choose the right parametric model that overfits the data less than the more flexible, nonparametric models. Here a good idea is to exploit emerging situations of multi-task learning, where the similarity between tasks is captured by a common prior that can be assumed in a nonparametric form. The present state-of-art is that model identification is largely a trial-and-error methodology, where methods are chosen based mainly on experimental experience, and rarely on a systematic search. We should use domain knowledge in the process, but intuitions about what is, or is not important for the task, are often misleading, and we should always verify if the emerging hypotheses agree with what is observed in the data. An open issue is, to what extent the process of model identification can be automated (automatic methods exist for the simplest situations, such as the automatic feature selection in linear models with few predictors). In the Netflix contest automatic parameter tuning was extensively used by many teams – but it stays an open question, whether it is possible to automatically test a large group of effects, or a wide class of possible hidden structures. In the machine learning community some advocate approaches closer to the fully Bayesian one, imposing prior on all possible models, but such approaches are rather impractical. It is better to gradually, as experiments are performed and the data becomes better understood, to reject implausible models, and to limit the class of considered models. A more time-efficient framework for initial model identification, than working with fully specified probabilistic models, seem to be neural networks (even after almost three years of the Netflix contest, a large majority of methods in the most accurate ensembles were based on neural-networks-like gradient descent optimization of regularized cost functions).

- Implementation, 10% efforts – choosing the right method optimizing the parameters with respect to the determined evaluation criterion. An example can be the matrix factorization model in the Netflix task. Knowing the general plausible form of the model $c_i + d_j + \mathbf{u}_i^T \mathbf{v}_j$, questions appear, whether to use an approximate Bayesian

method, such as MCMC with Gibbs sampling or Variational Bayes, or to use the simple MAP estimation (which has not worked well for the variables u_{ik} multiplied by v_{jk}), or, instead of defining a probabilistic model, to decide on simpler algorithms in a neural networks approach, with proper regularization (nonstandard, linear regularization is needed for the SVD model in the Netflix task). Other questions are whether to learn the parameters one at a time or whole groups of parameters, and whether to optimize the appearing numerical criterion using a first-order method like gradient descent (stochastic or batch learning, the momentum method, or perhaps conjugate gradient), or a second-order method, by jumping directly to the minimum (of one parameter or of a group of parameters) in case of a quadratic cost function. Similar questions appear when implementing models for other datasets and tasks.

The listed three phases may alternate. Findings from subsequent phases may help in the earlier ones, perhaps leading to redefining the task and the evaluation method. The whole process of data analysis may need to be repeatedly iterated.

The prediction contest ended by crossing the arbitrary level of 10% accuracy improvement over the reference algorithm, but questions remain: what is the best accuracy possible to attain? Which probabilistic model generated the data? Which dimensionality reduction method is the right one (inside the unknown, optimal model)? Despite the remaining questions, the overall conclusions should hold. To model various aspects of the real world on the basis of gathered data it is useful and often inevitable to use methods and approaches similar to the described in this work. Experiences with tasks like the Netflix Prize are part of the journey approaching the optimal process of data analysis.

References

[Ada11] Panagiotis Adamopoulos, Alexander Tuzhilin. *On Unexpectedness in Recommender Systems: Or How to Expect the Unexpected*, RecSys, 2011.

[Ado10] Gediminas Adomavicius, Jingjing Zhang. *On the stability of recommendation algorithms*, RecSys, 2010.

[Ado11] Gediminas Adomavicius, Jesse Bockstedt, Shawn Curley, Jingjing Zhang. *Recommender Systems, Consumer Preferences, and Anchoring Effects*, Workshop on Human Decision Making in Recommender Systems, RecSys, 2011.

[Aga09] Deepak Agarwal, Bee-Chung Chen. *Regression-based Latent Factor Models*, KDD, 2009.

[Aga10] Deepak Agarwal, Bee-Chung Chen. *fLDA: Matrix Factorization through Latent Dirichlet Allocation*, WSDM, 2010.

[Ali04] Kamal Ali, Wijnand van Stam. *TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture*, KDD, 2004.

[All10] Genevera I. Allen, Robert Tibshirani. *Transposable Regularized Covariance Models with an Application to Missing Data Imputation*, The Annals of Applied Statistics, 2010.

[Ama09] Xavier Amatriain, Nuria Oliver, Josep M. Pujol, Nava Tintarev. *Rate it Again: Increasing Recommendation Accuracy by User re-Rating*, RecSys, 2009.

[Ama12] Xavier Amatriain, Justin Basilico. *Netflix Recommendations: Beyond the 5 stars*, The Netflix Tech Blog, http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html, 2012.

[Bak03] Bart Bakker, Tom Heskes. *Task Clustering and Gating for Bayesian Multitask Learning*, JMLR 2003.

[Bel07a] Robert M. Bell, Yehuda Koren. *Modelling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems*, KDD, 2007.

[Bel07b] Robert M. Bell, Yehuda Koren. *Improved Neighborhood-based Collaborative Filtering*, Proc. KDD Cup and Workshop, 2007.

[Bel07c] Robert M. Bell, Yehuda Koren, Chris Volinsky. *The BellKor solution to the Netflix Prize*, 2007.

[Bel07d] Robert M. Bell, Yehuda Koren. *Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights*, ICDM, 2007.

[Bel07e] Robert M. Bell, Yehuda Koren. *Lessons from the Netflix Prize Challenge*, SIGKDD Explorations, 2008.

[Bel07f] Robert M. Bell, Yehuda Koren, Chris Volinsky. *Chasing $1,000,000: How we won the Netflix Progress Prize*, Statistical Computing & Graphics, 2007.

[Bel08] Robert M. Bell, Yehuda Koren, Chris Volinsky. *The BellKor 2008 Solution to the Netflix Prize* 2008.

[Ben07] James Bennett, Stan Lanning. *The Netflix Prize*, Proc. KDD Cup and Workshop, 2007.

[Bic07] Steffen Bickel, Michael Bruckner, Tobias Scheffer. *Discriminative Learning for Differing Training and Test Distributions*, ICML 2007.

[Bic09] Steffen Bickel, Michael Brückner, Tobias Scheffer. *Discriminative Learning Under Covariate Shift*, JMLR 10, 2009.

[Bil98] Daniel Billsus, Michael J. Pazzani. *Learning Collaborative Information Filters*, ICML, 1998.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*, Springer, 2006.

[Bis99a] Christopher M. Bishop. *Bayesian PCA*, NIPS, 1999.

[Bis99b] Christopher M. Bishop. *Variational Principal Components*, International Conference on Artificial Neural Networks, 1999.

[Bla92] Fischer Black, Robert Litterman. *Global Portfolio Optimization*, Financial Analysis Journal, 1992.

[Ble03] David M. Blei, Andrew Y. Ng, Michael I. Jordan. *Latent Dirichlet Allocation*, JMLR 3, 2003.

[Bra02] Matthew Brand. *Incremental singular value decomposition of uncertain data with missing values*, European Conference on Computer Vision, 2002.

[Bra03] Matthew Brand. *Fast online SVD revisions for lightweight recommender systems*, SDM, 2003.

[Bre71] Richard P. Brent. *Algorithms for Finding Zeros and Extrema of Functions Without Calculating Derivatives*, PhD thesis, 1971.

[Bri39] Willard C. Brinton. *Graphic presentation*, 1939.

[Bry07] E. Brynjolfsson, Y.J. Hu, D. Simester. *Goodbye Pareto Principle, Hello Long Tail: The Effect of Search Costs on the Concentration of Product Sales*, 2005-2007.

[Bur02] Robert Burke. *Hybrid Recommender Systems: Survey and Experiments.* User Modelling and User-Adapted Interaction, 2002.

[Bur05] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, Greg Hullender. *Learning to Rank using Gradient Descent*, ICML, 2005.

[Cai08] Jian-Feng Cai, Emmanuel J. Candes, Zuowei Shen. *A Singular Value Thresholding Algorithm for Matrix Completion*, 2008.

[Can02] John Canny. *Collaborative Filtering with Privacy via Factor Analysis*, SIGIR, 2002.

[Can09] Emmanuel J. Candes, Terence Tao. *The Power of Convex Relaxation: Near-Optimal Matrix Completion*, 2009.

[Car00] Bradely P. Carlin, Thomas A. Louis. *Bayes and empirical Bayes methods for data analysis*, Chapman and Hall, 2000.

[Car97] Rich Caruana. *Multitask learning*, PhD Thesis, 1997.

[Cha05] Oliver Chapelle, Zaid Harchaoui. *A Machine Learning Approach to Conjoint Analysis*, NIPS, 2005.

[Cha11] Gil Chamiel. *Utilising Structured Information for the Representation and Elicitation of User Preferences*, PhD thesis, 2011.

[Cha97] Daniel Chandler. *An Introduction to Genre Theory*, 1997.

[Che11a] Po-Lung Chen, et al.. *A Linear Ensemble of Individual and Blended Models for Music Rating Prediction*, KDD Cup and Workshop, 2011.

[Che11b] Tianqi Chen, Zhao Zheng, Qiuxia Lu, Weinan Zhang, Yong Yu. *Feature-Based Matrix Factorization*, Technical report, 2011.

[Cho11] Sean Choi, Ernest Ryu, Yuekai Sun. *Yelp++ : 10 Times More Information per View*, 2011.

[Cic09] Andrzej Cichocki — Rafał Zdunek, Anh Huy Phan, Shun-ichi Amari. *Nonnegative Matrix and Tensor Factorizations. Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*, Wiley, 2009.

[Dau06] Hal Daumé III, Daniel Marcu. *Domain Adaptation for Statistical Classifiers*, Journal of Artificial Intelligence Research, 2006.

[Dau07] Hal Daume III. *Frustratingly Easy Domain Adaptation*, ACL, 2007.

[DeC06] Dennis DeCoste. *Collaborative Prediction Using Ensembles of Maximum Margin Matrix Factorizations*, ICML, 2006.

[Del08] Nicolas Delannay, Michel Verleysen. *Collaborative filtering with interlaced generalized linear models*, Neurocomputing, 2008.

[Dro11] Gideon Dror, Noam Koenigstein, Yehuda Koren, Markus Weimer. *The Yahoo! Music Dataset and KDD-Cup'11* KDD Cup, 2011.

[Far04] Julian Faraway. *Linear Models with R*, CRC Press, 2004.

[Far06] Julian Faraway. *Extending the Linear Model with R*, CRC Press, 2006.

[Faz01] Maryam Fazel, Haitham Hindi, Stephen P. Boyd. *A Rank Minimization Heuristic with Application to Minimum Order System Approximation*, American Control Conference, 2001.

[Faz03] Maryam Fazel, Haitham Hindi, Stephen P. Boyd. *Log-det heuristic for matrix rank minimization with applications to Hankel and Euclidean distance matrices*, American Control Conference, 2003.

[Fre11] Christoph Freudenthaler, Lars Schmidt-Thieme, Steffen Rendle. *Bayesian Factorization Machines*, 2011.

[Fun06] Simon Funk. *Netflix Update: Try This at Home*, http://sifter.org/ simon/journal/20061211.html, 2006.

[Fun07] Simon Funk. *Netflix Reprize*, http://sifter.org/ simon/journal/20070817.html, 2007.

[Gan09] Emden Gansner, Yifan Hu, Stephen Kobourov, Chris Volinsky. *Putting Recommendations on the Map – Visualizing Clusters and Relations*, RecSys, 2009.

[Gauss1809] Johann Carl Friedrich Gauss. *Theoria Motus Corporum Coelestium in sectionibus conicis solem ambientium (Theory of the motion of the heavenly bodies moving about the sun in conic sections)*, 1809, English transl. by Charles H. Davis 1857.

[Gel11] Andrew Gelman, Cosma Rohilla Shalizi. *Philosophy and the practice of Bayesian statistics*, 2011.

[Gel95] Andrew Gelman, Donald B. Rubin. *Avoiding model selection in Bayesian social research*, Sociological Methodology, 1995.

[Gil10] Nicolas Gillis, Francois Glineur1. *Low-Rank Matrix Approximation with Weights or Missing Data is NP-hard*, JMLR, 2010.

[Gle06] David Gleich, Matthew Rasmussen, Kevin Lang, Leonid Zhukov. *The World of Music: User Ratings; Spectral and Spherical Embeddings; Map Projections*, 2006.

[Gli99] Mark E. Glickman. *Parameter estimation in large dynamic paired comparison experiments*, Journal of the Royal Statistical Society: Series C (Applied Statistics) Vol. 48, Issue 3, 1999.

[Goe10] Sharad Goel, Andrei Broder, Evgeniy Gabrilovich, Bo Pang. *Anatomy of the Long Tail: Ordinary People with Extraordinary Tastes*, WSDM 2010.

[Gol01] Ken Goldberg, Theresa Roeder, Dhruv Gupta, Chris Perkins. *Eigentaste: A Constant Time Collaborative Filtering Algorithm*, Information Retrieval, 2001.

[Gol92] David Goldberg, David Nichols, Brian M. Oki, Douglas . *Using collaborative filtering to weave an information tapestry*, Communications of the ACM, 1992.

[Har07] Paul Harrison. *How to get an RMSE of 0.8937 in the NetFlix Challenge*, http://logarithmic.net/pfh/blog/01176798503, 2007.

[Har11] Morgan Harvey, Mark J. Carman, Ian Ruthven, Fabio Crestani. *Bayesian Latent Variable Models for Collaborative Item Rating Prediction*, CIKM, 2011.

[Has09] Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition*, Springer, 2009.

[Has10] Shahzaib Hassan, Zeeshan Syed. *From netflix to heart attacks: collaborative filtering in medical datasets*, ACM International Health Informatics Symposium, 2010.

[Hav97] Håvard Rue and Øyvind Salvesen. *Predicting and retrospective analysis of soccer matches in a league*, 1997.

[Her00a] Jonathan L. Herlocker. *Understanding and Improving Automated Collaborative Filtering Systems*, PhD thesis, University of Minnesota, 2000.

[Her00b] Jonathan L. Herlocker, Joseph A. Konstan, John Riedl. *Explaining Collaborative Filtering Recommendations*, ACM Conference on Computer Supported Cooperative Work, 2000.

[Her04] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, John T. Riedl. *Evaluating Collaborative Filtering Recommender Systems*, ACM Transactions on Information Systems 2004.

[Her07] Ralf Herbrich, Thore Graepel. *TrueSkill: A Bayesian skill rating system*, NIPS, 2007.

[Hid12] Balazs Hidasi, Domonkos Tikk. *Enhancing Matrix Factorization Through Initialization for Implicit Feedback Databases*, Workshop on Context-awareness in Retrieval and Recommendation, 2012.

[Hil95] Will Hill, Larry Stead, Mark Rosenstein, George Furnas . *Recommending And Evaluating Choices In A Virtual Community Of Use*, SIGCHI Conference on Human Factors in Computing Systems, 2005.

[Hin02] Geoffrey E. Hinton. *Training products of experts by minimizing contrastive divergence*, Neural Computation, 2002.

[Hin10] Geoffrey Hinton. *A Practical Guide to Training Restricted Boltzmann Machines*, Technical Report, 2010.

[Hin95] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, Radford M Neal. *The wake-sleep algorithm for unsupervised neural networks*, Science, 1995.

[Hof03] Thomas Hofmann. *Collaborative Filtering via Gaussian Probabilistic Latent Semantic Analysis*, SIGIR, 2003.

[Hof04] Thomas Hofmann. *Latent Semantic Models for Collaborative Filtering*, ACM Transactions on Information Systems, 2004.

[Hof99a] Thomas Hofmann, Jan Puzicha, Michael I. Jordan. *Learning from Dyadic Data*, NIPS, 1999.

[Hof99b] Thomas Hofmann. *Probabilistic Latent Semantic Analysis*, UAI, 1999.

[Hof99c] Thomas Hofmann. *Probabilistic Latent Semantic Indexing*, SIGIR, 1999.

[Hu10] Rong Hu, Pearl Pu. *A Study on User Perception of Personality-Based Recommender Systems*, User Modeling, Adaptation, and Personalization, 2010.

[Ili08] Alexander Ilin, Tapani Raiko. *Practical Approaches to Principal Component Analysis in the Presence of Missing Values*, TKK Reports in Information and Computer Science, 2008 (also JMLR 2010).

[Jag10] Martin Jaggi, Marek Sulovsk'y. *A Simple Algorithm for Nuclear Norm Regularized Problems*, ICML, 2010.

[Jah10] Michael Jahrer, Andreas Töscher, Robert Legenstein. *Combining Predictions for Accurate Recommender Systems*, KDD, 2010.

[Jah11a] Michael Jahrer, Andreas Toscher. *Collaborative Filtering Ensemble*, KDD Cup and Workshop, 2011.

[Jah11b] Michael Jahrer, Andreas Toscher. *Collaborative Filtering Ensemble for Ranking*, KDD Cup and Workshop, 2011.

[Jan10] Dietmar Jannach, Markus Zanker, Alexander Felfernig, Gerhard Friedrich. *Recommender Systems: An Introduction*, Cambridge University Press, 2010.

[Kag09] Martijn Kagie, Matthijs van der Loos, Michiel van Wezel. *Including Item Characteristics in the probabilistic Latent Semantic Analysis Model for Collaborative Filtering*, AI Communications, 2009.

[Kel56] J.L.Kelly. *A New Interpretation of Information Rate*, The Bell System Technical Journal, July 1956.

[Kho10] Mohammad Khoshneshin, W. Nick Street. *Collaborative Filtering via Euclidean Embedding*, RecSys, 2010.

[Kon97] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, John Riedl. *GroupLens: applying collaborative filtering to Usenet news*, Communications of the ACM, 1997.

[Kor06] Jacek Koronacki, Jan Mielniczuk. *Statystyka dla studentów kierunków technicznych i przyrodniczych*, WNT, 2006.

[Kor08] Yehuda Koren. *Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model*, KDD, 2008.

[Kor09a] Yehuda Koren. *Collaborative filtering with temporal dynamics*, KDD, 2009.

[Kor09b] Yehuda Koren. *The BellKor Solution to the Netflix Grand Prize*, 2009.

[Kor10] Yehuda Koren. *Factor in the Neighbors: Scalable and Accurate Collaborative Filtering*, TKDD, 2010.

[Kor11] Yehuda Koren, Joe Sill. *OrdRec: an ordinal model for predicting personalized item rating distributions*, RecSys, 2011.

[Kul09] Wojciech Kulik. *Framework for fast CF algorithms*, 2009.

[Kur07] Miklos Kurucz, Andras A. Benczur, Karoly Csalogany. *Methods for large scale SVD with missing values*, Proc. KDD Cup and Workshop, 2007.

[Kwo11] YoungOk Kwon. *Computational Techniques For More Accurate and Diverse Recommendations*, PhD thesis, 2011.

[Lai11] Siwei Lai, Liang Xiang, Rui Diao, Yang Liu, Huxiang Gu, Liheng Xu, Hang Li, Dong Wang, Kang Liu, Jun Zhao, Chunhong Pan. *Hybrid Recommendation Models for Binary User Preference Prediction Problem*, KDD Cup and Workshop, 2011.

[Lam04] Shyong K. Lam, John Riedl. *Shilling Recommender Systems for Fun and Profit*, WWW, 2004.

[Lat10] Neal Lathia, Stephen Hailes, Licia Capra, Xavier Amatriain. *Temporal Diversity in Recommender Systems*, SIGIR, 2010.

[Law03] Neil D. Lawrence. *Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data*, NIPS 2003.

[Law05] Neil Lawrence. *Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models*, Journal of Machine Learning Research, 2005.

[Law09] Neil D. Lawrence, Raquel Urtasun. *Non-linear matrix factorization with gaussian processes*, ICML, 2009.

[Lee00] Daniel D. Lee, H. Sebastian Seung. *Algorithms for Non-negative Matrix Factorization*, NIPS, 2000.

[Lee08] John Lees-Miller, Fraser Anderson, Bret Hoehn, Russell Greiner. *Does Wikipedia Information Help Netflix Predictions?*, ICML, 2008.

[Lem05] Daniel Lemire, Anna Maclachlan. *Slope One Predictors for Online Rating-Based Collaborative Filtering*, SDM, 2005.

[Lim07] Yew Jin Lim, Yee Whye Teh. *Variational Bayesian Approach to Movie Rating Prediction*, Proc. KDD Cup and Workshop, 2007.

[Lin03] Greg Linden, Brent Smith, Jeremy York. *Amazon.com recommendations: Item-to-item collaborative filtering*, IEEE Internet Computing 7, 2003.

[Liu09] Ji Liu, Przemyslaw Musialski, Peter Wonka, Jieping Ye. *Tensor Completion for Estimating Missing Values in Visual Data*, Computer Vision, 2009.

[Lon10] Philip M. Long, Rocco A. Servedio. *Restricted Boltzmann Machines are Hard to Approximately Evaluate or Simulate*, ICML, 2010.

[Ma08] Shiquian Ma, Donald Goldfarb, Lifeng Chen. *Fixed point and Bregman iterative methods for matrix rank minimization*, 2008.

[Mac03] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[Mac10] Lester Mackey, David Weiss, Michael I. Jordan. *Mixed Membership Matrix Factorization*, ICML, 2010.

[Mad03] M.R. Madruga, C.A.B. Pereira, J.M. Stern. *Bayesian evidence test for precise hypotheses*, Journal of Statistical Planning and Inference, 2003.

[Man82] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*, W. H. Freeman, 1982.

[Mar04] Benjamin Marlin. *Collaborative filtering: A machine learning perspective*, MSc thesis, 2004.

[Mar05] Benjamin Marlin, Sam Roweis, Richard Zemel. *Unsupervised Learning with Non-Ignorable Missing Data*, Workshop on Artificial Intelligence and Statistics, AISTAT, 2005.

[Mar08] Benjamin Marlin. *Missing data problems in machine learning*, PhD thesis, University of Toronto.

[Mar09] Benjamin M. Marlin and Richard Zemel. *Collaborative Prediction and Ranking with Non-Random Missing Data*, RecSys, 2009.

[McF12] Brian McFee, Thierry Bertin-Mahieux, Daniel P.W. Ellis, Gert R.G. Lanckriet. *Collaborative Prediction and Ranking with Non-Random Missing Data*, WWW, 2012.

[McK11] McKenzie et al.. *Novel Models and Ensemble Techniques to Discriminate Favorite Items from Unrated Ones for Personalized Music Recommendation*, KDD Cup and Workshop, 2011.

[McM96] Daniel W. McMichael. *Estimating Gaussian Mixture Models from Data with Missing Features*, Signal Processing and Applications, 1996.

[McS09] Frank McSherry, Ilya Mironov. *Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders*, KDD, 2009.

[Mee09] Lydia Meesters, Paul Marrow, Bart Knijnenburg, Don Bouwhuis, Maxine Glancy. *ICT MyMedia Project. Deliverable 1.5. End-user recommendation evaluation metrics*, ICT MyMedia, 2009.

[Meh09] Bhaskar Mehta, Wolfgang Nejdl. *Unsupervised strategies for shilling detection and robust collaborative filtering*, User Modeling and User-Adapted Interaction, 2009.

[Mey12] Frank Meyer. *Recommender systems in industrial contexts*, PhD thesis, 2012.

[Mil03] Bradley N. Miller, Istvan Albert, Shyong K. Lam, Joseph A. Konstan, John Riedl. *MovieLens Unplugged: Experiences with an Occasionally Connected Recommender Systems*, Intelligent User Interfaces, 2003.

[Mis11] B. Mishra, G. Meyer, R. Sepulchre. *Low-rank optimization for distance matrix completion*, IEEE Conference on Decision and Control, 2011.

[Mni10] Andriy Mnih. *Learning Distributed Representations for Statistical Language Modelling and Collaborative Filtering*, PhD thesis, 2010.

[Mni11] Andriy Mnih. *Taxonomy-informed latent factor models for implicit feedback*, KDD Cup and Workshop, 2011.

[Mon03] Miquel Montaner, Beatriz López, Josep Lluís de la Rosa. *A Taxonomy of Recommender Agents on the Internet*, Artificial Intelligence Review, 2003.

[Mos95] Klaus Mosegaard, Albert Tarantola. *Monte Carlo sampling of solutions to inverse problems*, Journal of Geophysical Research, 1995.

[Nak09] Shinichi Nakajima, Masashi Sugiyama. *Analysis of Variational Bayesian Matrix Factorization*, KDD, 2009.

[Nak10a] Shinichi Nakajima, Masashi Sugiyama. *Implicit Regularization in Variational Bayesian Matrix Factorization*, ICML, 2010.

[Nak10b] Shinichi Nakajima, Masashi Sugiyama, Ryota Tomioka. *Global Analytic Solution for Variational Bayesian Matrix Factorization*, NIPS, 2010.

[Nak11a] Shinichi Nakajima, Masashi Sugiyama, Derin Babacan. *On Bayesian PCA: Automatic Dimensionality Selection and Analytic Solution*, ICML, 2011.

[Nak11b] Shinichi Nakajima, Masashi Sugiyama. *Theoretical Analysis of Bayesian Matrix Factorization*, JMLR 12, 2011.

[Nar08] Arvind Narayanan and Vitaly Shmatikov. *Robust De-anonymization of Large Sparse Datasets*, IEEE Symposium on Security and Privacy, 2008.

[New03] M.E.J. Newman. *The structure and function of complex networks*, SIAM Review, 2003.

[New10] Chris Newell, Bart Knijnenburg. *ICT MyMedia Project. Deliverable 5.4. Enhanced Internet A/V Content*, ICT MyMedia, 2010.

[NF07] *Netflix Prize: Forum* http://netflixprize.com/community/ .

[Nor07] Ragnar Norberg. *Credibility Theory*, 2007.

[Pal08] Andrzej Palczewski. *Portfolio optimization - a practical approach*, 2008.

[Paq10] Ulrich Paquet, Blaise Thomson, Ole Winther. *Large-scale Ordinal Collaborative Filtering*, 1st Workshop on Mining the Future Internet, Future Internet Symposium, 2010.

[Pat07] Arkadiusz Paterek. *Improving Regularized Singular Value Decomposition for Collaborative Filtering*, Proc. KDD Cup and Workshop, 2007.

[Pea09] Judea Pearl. *Causal inference in statistics: An overview* Statistics Surveys, 2009.

[Per10] Patrick O. Perry, Art B. Owen. *A Rotation Test to Verify Latent Structure*, JMLR 11, 2010.

[Pha06] Hoang Pham (ed.). *Handbook of Engineering Statistics*, Springer, 2006.

[Pil09a] István Pilászy, Domonkos Tikk. *Computational Complexity Reduction for Factorization-Based Collaborative Filtering Algorithms* E-Commerce and Web Technologies, 2009.

[Pil09b] István Pilászy, Domonkos Tikk. *Recommending New Movies: Even a Few Ratings Are More Valuable Than Metadata* RecSys, 2009.

[Pil09c] István Pilászy, Domonkos Tikk. *Explaining Recommendations of Factorization-Based Collaborative Filtering Algorithms*, Acta Technica Jourinensis, 2009.

[Pil09d] István Pilászy. *Factorization-Based Large Scale Recommendation Algorithms*, PhD Thesis, 2009.

[Pil10] István Pilászy, Dávid Zibriczky, Domonkos Tikk. *Fast als-based matrix factorization for explicit and implicit feedback datasets*, RecSys, 2010.

[Pio09] Martin Piotte, Martin Chabbert. *The Pragmatic Theory solution to the Netflix grand prize*, 2009.

[Por08] Ian Porteous, Evgeniy Bart, Max Welling. *Multi-HDP: A Non Parametric Bayesian Model for Tensor Factorization*, AAAI, 2008.

[Por10a] Ian Porteous, Arthur Asuncion, Max Welling. *Bayesian Matrix Factorization with Side Information and Dirichlet Process Mixtures*, AAAI, 2010.

[Por10b] Ian Porteous. *Mixture Block Methods for Non Parametric Bayesian Models with Applications*, PhD Thesis, 2010.

[Pot08] Gavin Potter. *Putting the collaborator back into collaborative filtering*, Proc. KDD Workshop, 2008.

[Pra11] Bruno Pradel, Savaneary Sean, Julien Delporte, Sébastien Guérif, Céline Rouveirol, Nicolas Usunier, Françoise Fogelman-Soulié, Frédéric Dufau-Joel. *A Case Study in a Recommender System Based on Purchase Data*, KDD, 2011.

[Pry98] Michael H. Pryor. *The Effects of Singular Value Decomposition on Collaborative Filtering*, Senior Honors Thesis, 1998.

[Rai07] Tapani Raiko, Alexander Ilin, Juha Karhunen. *Principal Component Analysis for Large Scale Problems with Lots of Missing Values*, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, 2007.

[Ras03] Carl E. Rasmussen. *Gaussian Processes in Machine Learning*, Springer, 2003.

[Ras06] Carl E. Rasmussen, Christopher K. I. Williams. *Gaussian Processes for Machine Learning*, MIT Press, 2006.

[Ren05] Jason D.M.Rennie, Nathan Srebro. *Fast Maximum Margin Matrix Factorization for Collaborative Prediction*, ICML 2005.

[Ren09] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, Lars Schmidt-Thieme. *BPR: Bayesian Personalized Ranking from Implicit Feedback*, UAI, 2009.

[Res94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, John Riedl. *GroupLens: An Open Architecture for Collaborative Filtering of Netnews*, ACM Conference on Computer Supported Cooperative Work, 1994.

[Ric10] Eds: Francesco Ricci, Lior Rokach, Bracha Shapira, Paul Kantor. *Recommender Systems Handbook*, Springer, 2010.

[Rob10] William J. J. Roberts. *Application of a Gaussian, Missing-Data Model to Product Recommendation*, IEEE Signal Processing Letters, 2010.

[Row98] Sam Roweis. *EM Algorithms for PCA and SPCA*, NIPS, 1998.

[Row99] Sam Roweis, Zoubin Ghahramani. *A Unifying Review of Linear Gaussian Models*, Neural Computation, 1999.

[Sal07a] Ruslan Salakhutdinov, Andriy Mnih, Geoffrey Hinton. *Restricted Boltzmann Machines for Collaborative Filtering*, ICML, 2007.

[Sal07b] Ruslan Salakhutdinov, Andriy Mnih. *Probabilistic Matrix Factorization*, Advances in Neural Information Processing Systems, 2007.

[Sal08] Ruslan Salakhutdinov, Andriy Mnih. *Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo*, ICML, 2008.

[Sal09] Ruslan Salakhutdinov. *Learning Deep Generative Models*, PhD thesis, 2009.

[Sal10] Ruslan Salakhutdinov, Nathan Srebro. *Collaborative Filtering in a Non-Uniform World: Learning with the Weighted Trace Norm*, 2010.

[Sar00] B.M Sarwar, G. Karypis, Joseph A. Konstan, John Riedl. *Application of Dimensionality Reduction in Recommender System–A Case Study*, WebKDD workshop, 2000.

[Sar01] B.M Sarwar, G. Karypis, Joseph A. Konstan, John Riedl. *Item-Based Collaborative Filtering Recommendation Algorithm*, WWW, 2001.

[Sch02a] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, David M. Pennock. *Methods and Metrics for Cold-Start Recommendations*, SIGIR, 2002.

[Sch02b] J.B.Schafer, Joseph A. Konstan, John Riedl. *Meta-Recommendation Systems: User-Controlled Integration Of Diverse Recommendations* International Conference on Information And Knowledge Management, 2002.

[Sch05] Anton Schwaighofer, Volker Tresp, Kai Yu. *Learning Gaussian Process Kernels via Hierarchical Bayes*, ICML, 2005.

[Sch07] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. *Collaborative Filtering Recommender Systems*, The Adaptive Web, 2007.

[Sch99] J. Ben Schafer, Joseph Konstan, John Riedl. *Recommender Systems in E-Commerce*, ACM conference on Electronic commerce, 1999.

[Sel11] Joachim Selke, Wolf-Tilo Balke. *Extracting Features from Ratings: The Role of Factor Models*, 2011.

[Sha10] Hanhuai Shan, Arindam Banerjee. *Generalized Probabilistic Matrix Factorizations for Collaborative Filtering*, Technical Report, 2010.

[Shi00] Hidetoshi Shimodaira. *Improving predictive inference under covariate shift by weighting the log-likelihood function*, Journal of Statistical Planning and Inference, 2000.

[Sil09] Joseph Sill, Gabor Takacs, Lester Mackey, David Lin. *Feature-Weighted Linear Stacking*, 2009.

[Sin08] Ajit P. Singh, Geoffrey J. Gordon. *A Unified View of Matrix Factorization Models*, Machine Learning and Knowledge Discovery in Databases, 2008.

[Sin09] Ajit Paul Singh. *Efficient Matrix Models for Relational Learning*, PhD thesis, Carnegie Mellon University, 2009.

[Ski07] David Skillicorn. *Understanding Complex Datasets: Data Mining with Matrix Decompositions*, Chapman and Hall, 2007.

[Sre04] Nathan Srebro. *Learning with matrix factorizations*, PhD thesis, 2004.

[Sre05] Nathan Srebro, Jason D.M.Rennie, Tommi S.Jaakkola. *Maximum-Margin Matrix Factorization*, NIPS, 2005.

[Sta11] Marius Stanescu *Rating systems with multiple factors*, MSc thesis, 2011.

[Ste09] David Stern, Ralf Herbrich, Thore Graepel. *Matchbox: Large Scale Online Bayesian Recommendations*, WWW, 2009.

[Ste10a] Julie Steele, Noah Iliinsky. *Beautiful Visualization: Looking at Data through the Eyes of Experts; Chapter 9 by Todd Holloway* O'Reilly Media, 2010.

[Ste10b] Harald Steck. *Training and Testing of Recommender Systems on Data Missing Not at Random*, KDD, 2010.

[Ste95] G.W.Stewart. *Gauss, Statistics, and Gaussian Elimination*, Journal of Computational and Graphical Statistics, 1995.

[Sug07] Masashi Sugiyama, Matthias Krauledat, Klaus-Robert Muller. *Covariate Shift Adaptation by Importance Weighted Cross Validation*, JMLR 8, 2007.

[Sug08] Masashi Sugiyama, Taiji Suzuki, Shinichi Nakajima, Hisashi Kashima, Paul von Bunau. *Direct Importance Estimation for Covariate Shift Adaptation*, Annals of the Institute of Statistical Mathematics, 2008.

[Tak07a] Gabor Takacs, István Pilászy, Bottyan Nemeth, Domonkos Tikk. *On the Gravity Recommendation System* Proc. KDD Cup and Workshop, 2007.

[Tak07b] Gabor Takacs, István Pilászy, Bottyan Nemeth, Domonkos Tikk. *Major components of the Gravity Recommendation System* SIGKDD Explorations, 2007.

[Tak08a] Gabor Takacs, István Pilászy, Bottyan Nemeth, Domonkos Tikk. *Investigation of Various Matrix Factorization Methods for Large Recommender Systems* Proc. KDD Workshop, 2008.

[Tak08b] Gabor Takacs, István Pilászy, Bottyan Nemeth, Domonkos Tikk. *Matrix factorization and neighbor based algorithms for the netflix prize problem* RecSys, 2008.

[Tak08c] Gabor Takacs, István Pilászy, Bottyan Nemeth, Domonkos Tikk. *A Unified Approach of Factor Models and Neighbor Based Methods for Large Recommender Systems* Applications of Digital Information and Web Technologies, 2008.

[Tak09a] Gabor Takacs, István Pilászy, Bottyan Nemeth, Domonkos Tikk. *Scalable collaborative filtering approaches for large recommender systems* JMLR 10, 2009.

[Tak09b] Gábor Takács. *Convex polyhedron learning and its applications*, PhD Thesis, 2009.

[Tan09] Tom F. Tan, Serguei Netessine. *Is Tom Cruise Threatened? Using Netflix Prize Data to Examine the Long Tail of Electronic Commerce*, Working paper, 2009.

[Tar05] Albert Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*, SIAM, 2005.

[Tip99] Michael E. Tipping, Christopher M. Bishop. *Probabilistic Principal Components Analysis*, Journal of the Royal Statistical Society, 1999.

[Tom07] John Tomfohr. *Private communication*, 2007-2009.

[Tos08a] Andreas Toscher, Michael Jahrer. *Improved Neighborhood-Based Algorithms for Large-Scale Recommender Systems*, Proc. KDD Workshop, 2008.

[Tos08b] Andreas Toscher, Michael Jahrer. *The BigChaos Solution to the Netflix Prize*, 2008.

[Tos09] *The BigChaos Solution to the Netflix Grand Prize*. 2009.

[Tos10] Andreas Töscher, Michael Jahrer *Collaborative Filtering Applied to Educational Data Mining*, JMLR, 2010.

[Tre04] Volker Tresp, Kai Yu. *An introduction to nonparametric hierarchical Bayesian modelling with a focus on multi-agent learning*, Proc. Hamilton Summer School on Switching and Learning in Feedback Systems, 2004.

[Tuk77] John W. Tukey. *Exploratory Data Analysis*, Addison Wesley, 1977.

[Wei07] M. Weimer, A. Karatzoglou, Q. Le, A. Smola. *Cofi rank – Maximum Margin Matrix Factorization for Collaborative Ranking*, NIPS, 2007.

[Wol02] Stephen Wolfram. *A New Kind of Science*, Wolfram Media, 2002.

[Wu07] Mingrui Wu. *Collaborative Filtering via Ensembles of Matrix Factorizations*, Proc. KDD Cup and Workshop, 2007.

[Wu08] Jinlong Wu, Tiejun Li. *A Modified Fuzzy C-Means Algorithm for Collaborative Filtering*, Proc. KDD Workshop, 2008.

[Wu09] Jinlong Wu. *Binomial Matrix Factorization for Discrete Collaborative Filtering*, ICDM, 2009.

[Xia09] Liang Xiang, Qing Yang. *Time-dependent Models in Collaborative Filtering based Recommender System*, Web Intelligence and Intelligent Agent Technologies, 2009.

[Xio09] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, Jaime G. Carbonell. *Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization*, 2009.

[Xue07] Y. Xue, X. Liao, L. Carin, B. Krishnapuram. *Multi-task learning for classification with Dirichlet Process priors*, JMLR, 2007.

[Yu03] Kai Yu, Anton Schwaighofer, Volker Tresp, Wei-Ying Ma, HongJiang Zhang. *Collaborative Ensemble Learning: Combining Collaborative and Content-Based Information Filtering via Hierarchical Bayes*, UAI, 2003.

[Yu04] K. Yu, A. Schwaighofer, V. Tresp, W.-Y. Ma, H. Zhang. *A nonparametric bayesian framework for information filtering*, SIGIR, 2004.

[Yu05a] Kai Yu, Volker Tresp, Anton Schwaighofer. *Learning Gaussian Processes from Multiple Tasks*, ICML, 2005.

[Yu05b] Kai Yu, Volker Tresp. *Learning to Learn and Collaborative Filtering*, NIPS, 2005.

[Yu06a] Kai Yu, Wei Chu, Shipeng Yu, Volker Tresp, Zhao Xu. *Stochastic Relational Models for Discriminative Link Prediction*, NIPS, 2006.

[Yu06b] Shipeng Yu, Kai Yu, Volker Tresp, Hans-Peter Kriegel. *Collaborative Ordinal Regression*, ICML, 2006.

[Yu07a] Kai Yu, Wei Chu. *Gaussian Process Models for Link Analysis and Transfer Learning*, NIPS, 2007.

[Yu07b] Shipeng Yu, Volker Tresp, Kai Yu. *Robust Multi-Task Learning with t-Processes*, ICML, 2007.

[Yu09a] Kai Yu, Shenghuo Zhu, John Lafferty, Yihong Gong. *Fast Nonparametric Matrix Factorization for Large-scale Collaborative Filtering*, SIGIR, 2009.

[Yu09b] Kai Yu, John Lafferty, Shenghuo Zhu, Yihong Gong. *Large-scale Collaborative Prediction Using a Nonparametric Random Effects Model*, ICML, 2009.

[Yue07] Yisong Yue, Thomas Finley, Filip Radlinski, Thorsten Joachims. *A Support Vector Method for Optimizing Average Precision*, SIGIR, 2007.

[Zha05] Sheng Zhang, Weihong Wang, James Ford, Fillia Makedon, Justin Pearlman. *Using Singular Value Decomposition Approximation for Collaborative Filtering*, IEEE International Conference on E-Commerce Technology, 2005.

[Zha06] Sheng Zhang, Weihong Wang, James Ford, Fillia Makedon. *Learning from Incomplete Ratings Using Non-negative Matrix Factorization*, SDM, 2006.

[Zha07a] Yi Zhang, Jonathan Koren. *Efficient Bayesian Hierarchical User Modeling for Recommendation Systems*, SIGIR, 2007.

[Zha07b] Yi-Cheng Zhang, Matus Medo, Jie Ren, Tao Zhou, Tao Li, Fan Yang. *Recommendation model based on opinion diffusion*, Europhysics Letters, 2007.

[Zha07c] Yi-Cheng Zhang, Marcel Blattner, Yi-Kuo Yu. *Heat Conduction Process on Community Networks as a Recommendation Model*, Physical Review Letters, 2007.

[Zha09] Yi Zhang, Jiazhong Nie. *Probabilistic Latent Relational Model for Integrating Heterogeneous Information for Recommendation.* Technical Report, 2009.

[Zho08] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, Rong Pan. *Large-scale Parallel Collaborative Filtering for the Netflix Prize*, Algorithmic Aspects in Information and Management, 2008.

[Zho10a] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph R. Wakeling, Yi-Cheng Zhang. *Solving the apparent diversity-accuracy dilemma of recommender systems*, Proceedings of the National Academy of Sciences, 2010.

[Zho10b] Mingyuan Zhou, Chunping Wang, Minhua Chen, John Paisley, David Dunson, Lawrence Carin. *Nonparametric Bayesian Matrix Completion*, IEEE Sensor Array and Multichannel Signal Processing Workshop, 2010.

[Zhu08] Shenghuo Zhu, Kai Yu, Yihong Gong. *Stochastic Relational Models for Large-scale Dyadic Data using MCMC*, NIPS, 2008.

[Zie05] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, Georg Lausen. *Improving Recommendation Lists Through Topic Diversification*, WWW, 2005.

Abbreviations:

KDD - ACM SIGKDD Conference on Knowledge Discovery and Data Mining

ICDM - IEEE International Conference on Data Mining

ICML - International Conference on Machine Learning

JMLR - Journal of Machine Learning Research

NIPS - Neural Information Processing Systems Conference

RecSys - ACM Conference on Recommender Systems

SDM - SIAM International Conference on Data Mining

SIGIR - ACM Special Interest Group on Information Retrieval Conference

WSDM - ACM International Conference on Web Search and Data Mining

WWW - International World Wide Web Conference