Universität Konstanz
FB Informatik und Informationswissenschaft
Master-Studiengang Information Engineering

Masterarbeit

Social-aware Matrix Factorization for Recommender Systems

zur Erlangung des akademischen Grades eines
Master of Science (M.Sc.)

von

Daniel Weidele

Erstgutachter:	Jun.-Prof. Dr. Steffen Rendle
Zweitgutachter:	Prof. Dr. Daniel A. Keim
Einreichung:	April 2013

To the best parents for their unceasing support.

Abstract

We review and categorize early approaches of collaborative filtering, before moving towards social-aware matrix factorization models for rating prediction, which we will theoretically compare to each other and to the state of the art model SVD++. We derive a generic social-aware facorization model and show how to improve runtime complexities of social-aware matrix factorization models in general. Moreover we discuss various trust metrics to exploit social network information and propose the application of PageRank as a new alternative in this context. Finally we provide a practical evaluation of presented approaches.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Whenever users of large scale information systems reach their limits in quantitatively exploring the information space, it can become task of a recommender system to close the gaps between growing amounts of data on the one hand, and users restricted in time and attention on the other hand. However, most information systems intend to retrieve results with maximum accuracy, also beyond the user's awareness of its own needs. Recommender systems therefore use techniques of data mining and information retrieval to make suggestions to the user, conceivably taking context into account. Moreover they may overcome a user's anonymous state of knowledge.

In general, recommender systems deal with a set of *users* and *items* such as e.g. movies[1], books[2], music[3], etc. Typically each user rates a set of items by some values, e.g. on a scale 1 (worst rating) to 5 (best rating). Given these ratings it is now task of the recommender to predict the ratings for a user on her non-rated items and consequently recommend items to the user.
A variant is tag recommendation where users assign tags to items and in turn the recommender suggests tags for new items.

Categorization

Based on their fundamental structure Balabanović and Shoham[2] classify recommender systems into two main categories. *Collaborative filtering* approaches back on the performance of many users, such that rating predictions are commonly a product of given opinions of similar users. *Content-based* recommenders more go into the details of item characteristics by exploiting their properties, such that predictions are based on a user's past rating of similar items. As usual, there are also hybrid approaches where

[1] http://www.netflix.com
[2] http://www.amazon.com
[3] http://www.last.fm

Table 1.1: Comparison of general recommender system approaches.

	Benefits	**Drawbacks**
Collaborative Filtering	⊕ Independence of content: no user- or item-knowledge required ⊕ Relatively fast prediction with models	⊖ Training phase takes its time ⊖ New users and items are not trivial to handle ⊖ Critical mass of observations required to achieve good results
Content-based recommenders	⊕ New items can be treated equally ⊕/⊖ Over-specialization: no totally different things will be proposed	⊖ Features have to be chosen carefully: different items have to be distinguishable ⊖ New users are not trivial to handle

both paradigms extend each other to overcome restrictions or disadvantages. In this regard table 1.1 compares known benefits and drawbacks of both approaches.

As we can see from figure 1.1 collaborative filtering has raised a lot of scientific attention during the past years. The plot shows the number of hits when searching for the term 'Collaborative Filtering' in Google Scholar[1]. From the figure itself we can interprete that not only the Netflix prize[15] (2006 to 2009) is responsible for the rising interest in this area, but of course it motivated the development and bundling up of various techniques in the field of collaborative filtering.

Breese, Heckerman and Kadie [6] further divide collaborative filtering approaches into a memory-based (or heuristic-based) and a model-based class. *Memory-based* algorithms take into account the entire rating observations of similar users to directly compute the prediction as a product of these. Compared *model-based* methods learn from the observations to train a model in first place, such that rating predictions are then computed indirectly with it. Figure 1.2 outlines the different flow of the approaches.

Recently a lot of research has been done in the area of collaborative filtering using probabilistic models to predict ratings. Before getting in touch with these we will first motivate the task of rating prediction in

[1] `http://google.com/scholar`

Figure 1.1: Rising number of publications from 1980 to 2012, based on search results for *Collaborative Filtering* in Google Scholar.

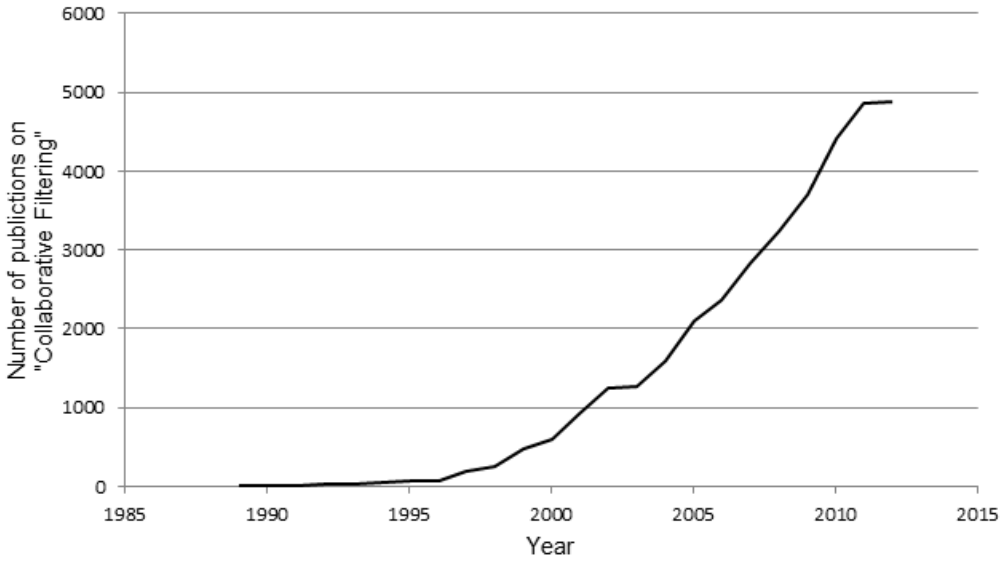
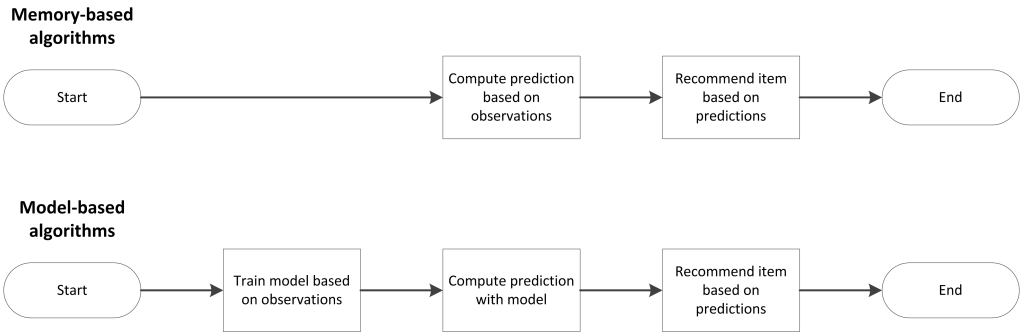


Figure 1.2: Comparison of typical flow in memory- and model-based collaborative filtering.



general (ch. 2) and gain an overview about various approaches experienced in the past (ch. 3), as it is advisable to know where we come from. In chapter 4 there is introduced a framework which constitutes the base for the utilization of social network information in rating prediction, as presented in chapter 5. We theoretically discuss social-aware matrix factorization models (ch. 6) and introduce various metrics for the incorporation of trust in chapter 7, before presenting practical experiences in the evaluation (ch. 8).

2 Motivation

2.1 Users, Items and Ratings

Today's online information systems offer access to millions of things to fairly every single user in the web. We have sites driven from the e-commerce domain offering products to sell or buy, as well as applications that support you in identifying your next vacation spot. There exist tools to compare car insurances, or web radio stations on which success or failure of your next commune party may depend on. Although the domains of these various systems seem to be quite different at first glance they are equal in a trivial way: they deal with domain-specific things and expose them to users. Let's call these things *items* of an information system, which allows *users* to interact in any domain-specific way (e.g. listening, buying or choosing). With most information systems allowing interaction with thousands of items on the one hand, but on the other hand users typically only wanting to face the item subset they are interested in, there exists a sort of natural need to filter and reduce the landscape of all items to finally receive the relevant subset. Classical information systems therefore offer controls, which allow users to input search terms, apply range queries, select by identifier and chiefly cut down in a *manual* way to reach their items of interest. However, a central task of recommender systems is the *automated* identification of items the user is expected to be most likely interested in.

With rising number of internet users, mobile devices and the ability to access information everywhere and anywhere, providers of many online applications recognize their opportunities opening up by collecting feedback from the users on the exposed items. Such feedback can be collected implicitly by evaluating accessed items by e.g. page views, or explicitly by facilitating to *rate* an item the user has interacted with. Imagine a couple of users, and some items that have partially been rated on a *rating scale* from e.g. 1 to 5, where 5 expresses like, and 1 expresses dislike about the item by the users. There are also some missing ratings, as typically not every single user has rated every single item, yet.

Table 2.1: Examplary rating matrix.

	Star Wars	**Titanic**	**Matrix**	**High Noon**
Alice		5	2	
Bob	5		4	
Christie	1	4		5
David	5	1	3	
Elvis	2			
Fred	2	3	5	2

For example we have a movie website (like IMDb[1]) where users are enabled to rate movies they have seen on a $[1, 5]$-scale as described above. Table 2.1 examplarily shows an obtained rating matrix of users *Alice*, *Bob*, *Christie*, *David* and *Elvis* for movies *Star Wars*, *Titanic* and *Matrix*, where empty cells denote missing ratings.

As this work concentrates on collaborative filtering it is based on such type of rating information, especially there is not required any detailed knowledge about properties of any item or user itself beyond an unqiue identifier (e.g. title or ID).

2.2 Rating Prediction

For item recommendation in recommender systems one option is to estimate the unknown ratings. Once the unknown ratings are available, a recommender could present top n items to the user, which are assumed to achieve the highest ratings from the user. From this point of view *rating prediction* is the essential task of a recommender system and by this represents the central goal to solve for this work. One could assume that users are most interested in items that they would rate highest, however, there is not made a statement on the correlation of *interestingness* and *rating value*, because low or outlying ratings could also be meaningful in the users perception.

Provided that highly rated items *are* most relevant, the predictions for user *Elvis* as shown in table 2.2 would lead to the recommendation of *High Noon* before *Titanic* followed by *Matrix* descendingly according to the predicted ratings. We keep in mind that no matter how predictions are further digested, accurate prediction of ratings stays the crucial part in such a recommendation process.

[1] http://imdb.com

Table 2.2: Examplary rating matrix after prediction for Elvis.

	Star Wars	**Titanic**	**Matrix**	**High Noon**
Alice		5	2	
Bob	5		4	
Christie	1	4		5
David	5	1	3	
Elvis	2	**3**	**1**	**5**
Fred	2	3	5	2

The task of rating prediction can moreover be formulated as a problem of *classification* or *regression*. In terms of classification one would regard each rating value as a class $c \in C$ and predict the most probable class given user u and item i. From a probabilistic view we choose

$$\mathcal{R}(u,i) = \arg\max_{c \in C} P(c|u,i) \tag{2.1}$$

as predicted rating $\mathcal{R}(u,i)$.
When seen as a regression problem, predicted ratings take shape of continuous values produced by a function of user u and item i:

$$\mathcal{R}(u,i) = f(u,i) \tag{2.2}$$

with f the desired or assumed rating function.

2.3 Social Interactions

In addition to the presented rating information today's information systems are furthermore enabled to collect context information which can be exploited for the task of rating prediction. Karatzoglou et al. present a generic way [17] to incorporate context information such as time of day, location or mood of the user about the rating event. They extend the 2-dimensional rating matrix by c additional dimensions (1 per context variable) to receive a $(c+2)$-dimensional tensor, for which again a regression problem with context variables $v_1, ..., v_c$

$$\mathcal{R}(u,i,v_1,...,v_c) = f(u,i,v_1,...,v_c) \tag{2.3}$$

has to be solved.

In this work we would like to consider *social interactions* among users, e.g. friendship or trust. These relations can be denoted as an adjancency matrix between users u and v with $(u, v) > 0$ if there exists a social relation between u and v, otherwise $(u, v) = 0$. Table 2.3 examplarily shows binary and undirected trust information among users.

Table 2.3: Adjacency matrix representing undirected social trust among users.

	Alice	**Bob**	**Christie**	**David**	**Elvis**	**Fred**
Alice	-	0	1	1	0	1
Bob	0	-	1	0	0	1
Christie	1	1	-	1	1	1
David	1	0	1	-	0	0
Elvis	0	0	1	0	-	1
Fred	1	1	1	0	1	-

Actually social relations among users are no kind of information that is specific to a single rating event, but it is equally valid over all ratings in the observation. Therefore we cannot simply introduce friendship as a new dimension in a rating tensor as presented in [17]. Nevertheless we will discover alternative methods to incorporate social information into rating prediction throughout this work, and evaluate whether it can improve prediction accuracy.

2.4 Contribution

The contribution of this work is versatile:

- First we review and categorize early approaches of collaborative filtering, before moving towards a probabilistic matrix factorization framework for rating prediction.
- Next there are presented related social-aware factorization models, which we will theoretically compare to each other and to the state of the art model SVD++.
- We further derive a generic social-aware facorization model and show how to improve runtime complexities in general.
- Moreover we discuss various trust metrics for the incorporation of social network information and propose PageRank as a new alternative.
- Finally we provide a practical evaluation of presented approaches.

3 A Retrospection of Collaborative Filtering

As presented in [1] there arose a whole bandwidth of approaches to fulfill the task of rating prediction and variants. In order to gain insight into various techniques and a more founded knowledge about recommender systems we will now have a detailed look at early (somehow historical) methods. However, we will mostly focus on collaborative filtering algorithms as this is more related to this work. In chapter 4 we progress to the matrix factorization model and also review a state of the art Netflix prize extension.

3.1 Nearest Neighborhood

First we will look at recommendation techniques, which identify *nearest neighbors* to predict with. In terms of collaborative filtering neighborhood can e.g. be determined by rating similarity.

PPMCC

For example, the GroupLens system[33] offers memory-based collaborative filtering to newsgroup participants and is based on a client-server architecture. Users can rate news articles via the client software. On the servers there are calculated predictions based on Pearson product-moment correlation coefficient (PPMCC) which means, that a rating prediction $\mathcal{R}(u,i)$ for user u and item i is explicitly modeled by correlations with other raters' taste. Let c_{uv} be the correlation coefficient between attitudes of users u and v, $R_{u,i}$ the rating of u for item i and U^i the set of all users that rated i. Then the proposed prediction works as follows:

$$\mathcal{R}(u,i) = \bar{R}_{u,\cdot} + \frac{\sum_{v \in U^i}(R_{v,i} - \bar{R}_{\cdot,i})c_{uv}}{\sum_{v \in U^i}|c_{uv}|} \tag{3.1}$$

with $\bar{R}_{\cdot,i}$ the mean rating of item i and

$$c_{uv} = \frac{\sum_{i \in I^u \cap I^v}(R_{u,i} - \bar{R}_{u,\cdot})(R_{v,i} - \bar{R}_{v,\cdot})}{\sqrt{\sum_{i \in I^u \cap I^v}(R_{u,i} - \bar{R}_{u,\cdot})^2 \sum_{i \in I^u \cap I^v}(R_{v,i} - \bar{R}_{v,\cdot})^2}} \tag{3.2}$$

where $\bar{R}_{u,\cdot}$ is the mean rating of user u.

TF-IDF

In contrast, content-based recommendation would more regard similarities of item content. The netnews-filtering system NewsWeeder[22] by Ken Lang uses the classic *term frequency–inverted document frequency* (TF-IDF) measure to convert each of the rating-categorized documents into feature vectors. TF-IDF is typically used in information retrieval to reflect the importance of terms/tokens t for document d given the document space D:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{3.3}$$

with $tf(t, d)$ the prevalence of t in d and $idf(t, D) = log\frac{|D|}{d_t}$, where d_t is the amount of documents containing t.

For each category there is calculated the average feature vector to be compared against new documents. The prediction is then based on the cosine similarity of new documents compared to these prototype vectors of each category. Given the document vector d and prototype vector p the cosine similarity can be derived as follows:

$$d \cdot p = ||d||_2 \; ||p||_2 \; cos\Theta \tag{3.4}$$

$$cos\Theta = \frac{d \cdot p}{||d||_2 \; ||p||_2} \tag{3.5}$$

Furthermore Lang proposes to use a minimum description length[36] approach and decide whether to view tokens as category-dependend or not in order to further adjust the weights.

IUF & Case Amplification

In [6] Breese et al. generalize equation 3.1 to

$$\mathcal{R}_{u,i} = \bar{R}_{u,\cdot} + k \sum_{v \in U \setminus \{u\}} w_{uv}(R_{v,i} - \bar{R}_{\cdot,i}) \tag{3.6}$$

where w_{uv} denotes any distance, correlation (eq. 3.2) or similarity (eq. 3.5) function among users and k is a normalizing factor. Since these functions can only iterate over the limited number of items, which have been rated by both users u and v, the authors propose *default voting*: for items only rated by one (or even none) of two users, they assume a default rating to fill in the missing value.

Moreover the authors transfer the TF-IDF measure (eq. 3.3), where IDF reduces weights for commonly occurring words, to *inverse user frequency*, such that uncommon items are cosidered more relevant in correlations. So in analogy, they define

$$iuf(i, U) = log\frac{|U|}{u_i} \tag{3.7}$$

where u_i is the number of all users that rated i. For e.g. cosine similarity, this can be applied by using a transformed rating $R'_{u,i}$ as the original rating $R_{u,i}$ multiplied by $iuf(i, U)$.

Furthermore the generalized equation 3.6 can be modified by what the authors call *case amplification*. Therefore they transform weights to

$$w'_{uv} = \begin{cases} w^{\rho}_{uv} & \text{, if } w_{uv} \geq 0 \\ -(-w^{\rho}_{uv}) & \text{, else} \end{cases} \tag{3.8}$$

with e.g. $\rho = 2.5$, in order to accentuate higher and lower weights.

Sample Application

Hill et al.[14] describe the setup of an early recommender system where people rate and receive predictions via e-mail. The focus of this work is not the proposal of a prediction technique, however, they reinforce the need for collaborative filtering in general. They work out that prediction quality significantly increases when recommenders take advantage of their so called *Virtual Communities*, as users tend to minimize effort in providing ratings or information about relations among each other (sparseness), while at the same time wanting to get the most out of the system. Virtual Communities represent loosely connected structures where users are cluelessly part of, and from which users may benefit as the system detects correlations and exploits these to the benefit of prediction quality.

Further Variants and Qualitative Results

The domain of music artists and albums is tackled by the Ringo system of Shardanand and Maes[39]. Users are allowed to rate artists via e-mail and may receive replies in the following three ways:

- List of artists that the user would probably like
- List of artists that the user would rather not like
- Rating prediction for a given artist and the user

Within the time of three months Ringo grew a community of 2100 users and processed 500 messages per day dealing with 3000 artists and 9000 albums. Equipped with (a subset of) these observations Shardanand and Maes split a source set R^S and a target set R^T to evaluate five prediction algorithms with respect to the mean absolute error

$$\bar{e} = \frac{1}{|R^T|} \sum_{(u,i) \in R^T} |R_{u,i} - \mathcal{R}(u,i)| = \frac{1}{|R^T|} \sum_{(u,i) \in R^T} |\mathcal{P}(u,i)| \tag{3.9}$$

and the standard deviation of the errors

$$\sigma = \sqrt{\frac{1}{|R^T|} \sum_{(u,i) \in R^T} (\mathcal{P}(u,i) - \bar{\mathcal{P}})^2} \tag{3.10}$$

which we will have a brief look at.

The *base case algorithm* simply predicts the mean score per artist received by R^S. It scores $\bar{e} = 1.3$ and $\sigma = 1.6$ for total R^T, but when focussing on the low and high ratings of R^T (polarizing items to love or hate) the situation changes. Figure 3.1 shows σ for the extrema of R^T (black) and the full set (white). In case of the extrema the lack of the missing bell curve (in fact there emerge two) suggests that the base case algorithms is not performing well.

The *mean squared differences algorithm* calculates the dissimilarity ρ_{uv} between two user rating vectors $R^b_{u,\cdot}, R^b_{v,\cdot}$ with number of b_{uv} artists not rated by both users are blacked out:

$$\rho_{uv} = \frac{(R^b_{u,\cdot} - R^b_{v,\cdot})^2}{|R^S| - b_{uv}} \tag{3.11}$$

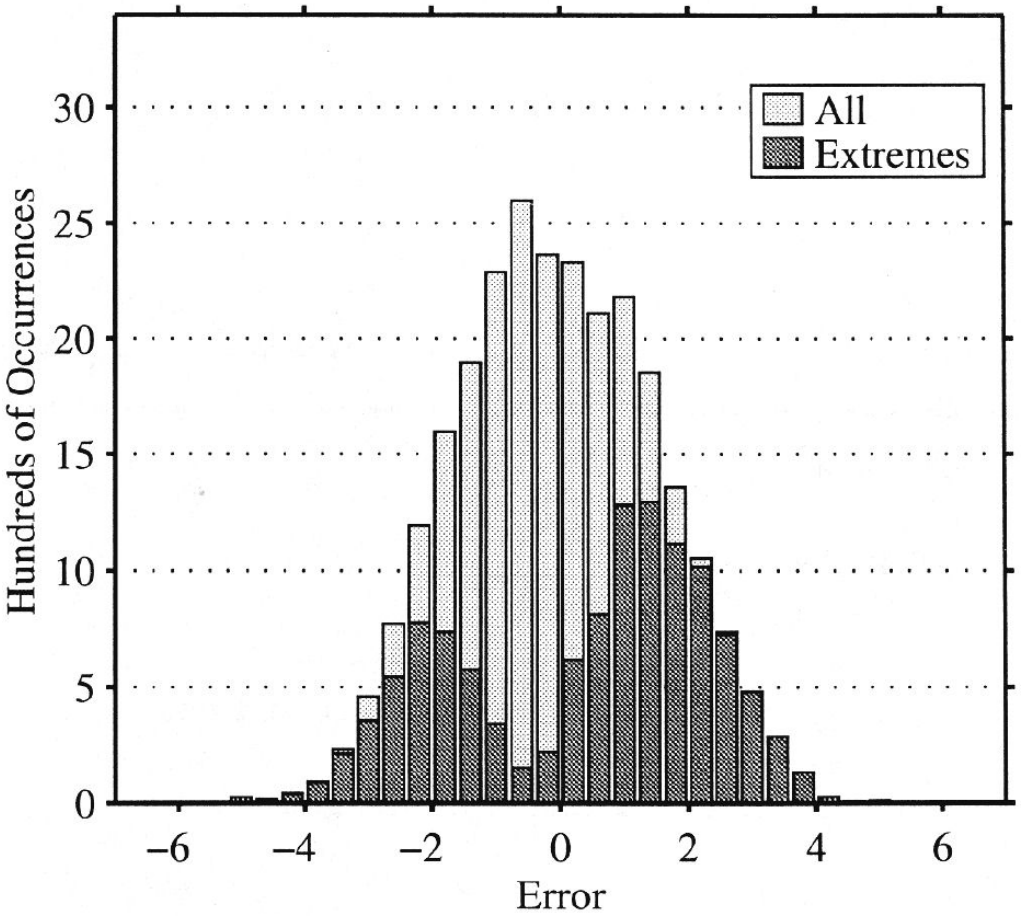
After dropping candidates with dissimilarities lower than a threshold Φ the weight w_{uv} of neighbor v to u is mapped to the interval $[0;1]$ by

$$w_{uv} = \frac{\Phi - \rho_{uv}}{\Phi} \tag{3.12}$$

such that the final rating prediction can be averaged over the set N_u of the neighbors of u:

$$\mathcal{R}(u,i) = \frac{\sum_{(v,j) \in R^S_{u,i}} w_{uv} R_{v,i}}{\sum_{(v,j) \in R^S_{u,i}} w_{uv}} \tag{3.13}$$

Figure 3.1: Errors of base case algorithm in the Ringo system.



with

$$R_{ui}^{S} = \{(v, j) \in R^{S} | v \in N_u, j = i\}$$

Furthermore Shardanand and Maes evaluate weighting with PPMCC, which we already looked at in eq. 3.2, as well as a variant of it where they fix both profile means within the correlation to the neutral rating value (e.g. 3 on the scale from 1 to 5) in order to align profile similarities with the general rating tendency.

Lastly the authors propose to apply an *artist-artist alrgotihm* where they do not correlate users but artists to weight the predictive influence.

Summarized for all algorithms $\bar{e}_{all}$ is between 1.0 and 1.3, σ_{all} between 1.3 and 1.6 — compared $\bar{e}_{extrema}$ is between 1.1 and 1.8, $\sigma_{extrema}$ between 1.5 and 2.0.

3.2 Classification

As outlined in section 2.2 rating prediction can furthermore be seen as a classification task. We will have a look at selected contributions, which are motivated from this direction.

Information Gain

Compared to the already mentioned mail-based application using Virtual Communities, there is provided more comfort (in terms of usability) by Pazzani and Billsus with their browser-based web page recommender 'Syskill & Webert'[28]. Here every web page is equipped with an extra header area, which allows the user to add the page to her *hot list* or *cold list* by a single click. Predictions are displayed as 'thumbs up/down' for every link on the web site, so that the user is supported in deciding where to navigate next. Given the hot and cold list, the system internally determines the most relevant words taken over all pages D by considering the expected information gain of each word w (stopwords are ignored) towards classification into classes $C = \{hot, cold\}$:

$$E(w, D) = H(D) - P(w)H(D_w^+) - (1 - P(w))H(D_w^-) \tag{3.14}$$

with P(w) the probability of w present on a page, D_w^+ (D_w^-) denoting the set of all pages containing (not containing) word w, and

$$H(D) = -\sum_{c \in C} P(c) log_2(P(c))$$

the entropy of classes with respect to D, where $P(c)$ is the probability of a document being classified as c.

Based on the so determined list of words, for each web page p there is extracted a boolean feature vector $(X_1, ...X_{|W|})$ indicating the presence or absense of a word w in page p. Given these feature vectors (and an independence assumption among words) the system models naïve Bayesian classification to predict the most likely class with respect to

$$\arg\max_{c \in C} P(c) \prod_{n \in |W|} P(X_n|c) \tag{3.15}$$

For the presented method the authors report an accuracy of 0.63 (topic: Goats) to 0.82 (topic: Sheep) evaluated by using 20 pages for training and the remaining ones (6 to 134 depening on the topic) for testing. The authors

also checked varying numbers (from 16 to 400) of considered words: they report a standard deviation $\sigma = 0.0054$ and best results (accuracy 0.76) with 96 features for the average of 6 topics.

Weighted Word Frequencies

Compared to the last method Mooney and Roy extend their book recommender LIBRA[26] by allowing the user to rate from 1 to 10 instead of simple 'thumbs up/down', such that the classification problem expands to 10 categories. Furthermore they use set-valued features[10] in order to profile items more precisely with respect to author, title, abstract, etc. To learn profiles, they internally also operate a naïve Bayesian classifier on boolean features of words, where they distinguish between same words among different set-attributes, e.g. $word_{title} ! = word_{abstract}$. For rating prediction, as there has to be calculated a more precise rating (1-10 instead of hot or cold), the authors propose to predict the sum among all categories

$$\mathcal{R}(b) = \sum_{c=1}^{10} cP(c|b) \tag{3.16}$$

with $P(c|b)$ the posterior probability of category c and $\mathcal{R}(b)$ the predicted rating of book b given a learned profile.
Additionally, from the above [1,10]-rating categories the authors derive a *weighted binary rating*-model by min-max-normalization

$$w_c = \frac{c - min(R_{u,\cdot})}{max(R_{u,\cdot}) - min(R_{u,\cdot})} \tag{3.17}$$

as the weigth of each rating category c of a user, in order to rewrite the number of occurrences n for a word per rating as

$$n' = nw_c \quad [n' = n(1 - w_c)] \tag{3.18}$$

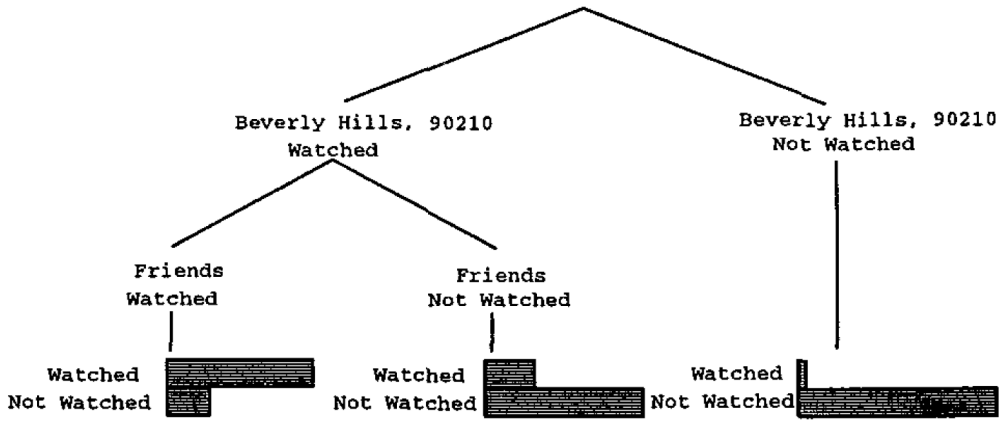
for positive [negative] ratings.

The result of their evaluation shows that the more training examples there are available, the better [1-10]-ratings and weighted binary ratings perform compared to simple binary classification. However, for smaller datasets (less than 900 examples) there is rather no significant difference among the approaches.

Bayesian Networks

Breese et al. introduce a Bayesian network model[6], where for each item there is created a node in the network. The states of the nodes represent rating values (including a state for 'not rated'). With the learning algorithm presented in [9] the authors identify the most predictive structure of the network, such that for each item they receive a set of parent nodes that best describe the conditional probabilities for the item. The parent nodes are used to build up a decision tree, which then encodes these conditional probabilities. Figure 3.2 shows an example of such a resulting decision tree for TV shows, where the probability for watching *Melrose Place* has been identified to depend on the show *Friends* and *Beverly Hills, 90210*. In the example there is only dealt with *watched* and *not watched* as rating values.

Figure 3.2: Decision tree encoding the conditional probabilities for having watched the TV show *Melrose Place*.



The evaluation shows that the performance of Bayesian networks improves more than nearest neighborhood approaches when learning at least 5 to 10 ratings per user, since the identified dependencies and probabilities optimize. For fewer ratings there cannot be observed a significant difference.

Personality Diagnosis

Pennock and Horvitz[29] motivate another collaborative filtering approach, where they assume that users rate items with Gaussian noise depending on context or the mood during the current session. Any rating $R_{u,i}$ of user u for item i is expected to be drawn from an independent normal distribution with mean $R_{u,i}^{true}$ and σ as a free parameter:

$$P(R_{u,i} = x | R_{u,i}^{true} = y) \propto e^{-(x-y)^2/2\sigma^2} \tag{3.19}$$

The prior probability that the active user a's *true* ratings are equal to a vector v is not explicitly counted from the observations, but defined to be a random rating vector of the n users with equal probability distribution $1/n$:

$$P(R_{a,\cdot}^{true} = R_{u,\cdot}) = \frac{1}{n} \tag{3.20}$$

By application of Bayes' rule equations 3.19 and 3.20 lead to the probability that the active user resembles any other user u's personality with respect to m items:

$$P(R_{a,\cdot}^{true} = R_{u,\cdot} | R_{a,1} = x_1, ..., R_{a,m} = x_m) \propto \tag{3.21}$$

$$P(R_{a,1} = x_1 | R_{a,1}^{true} = R_{u,1}) \cdots P(R_{a,m} = x_m | R_{a,m}^{true} = R_{u,m})$$

$$\cdot P(R_{a,\cdot}^{true} = R_{u,\cdot})$$

Based on this formula computed for each user u the predicted rating for an unseen item i is then defined as the probability

$$P(R_{a,i} = x_i | R_{a,1} = x_1, ..., R_{a,m} = x_m) = \tag{3.22}$$

$$\sum_{u \in U} P(R_{a,i} = x_i | R_{a,\cdot}^{true} = R_{u,\cdot})$$

$$\cdot P(R_{a,\cdot}^{true} = R_{u,\cdot} | R_{a,1} = x_1, ..., R_{a,m} = x_m)$$

From this calculation the *most probable* rating is returned as the predicted value. The authors further note that the model could be regarded as a clustering method with exactly one user per cluster, or as a diagnostic model with ratings as *symtpoms* and the probability of each personality type as the causing *disease*.
According to the empirical results Personality Diagnosis outperforms correlation-based approaches and Bayesian networks with respect to the mean absolute error using a dataset with 1623 items, 5000 users in training and 4119 users in a test set.

SVD & Artifical Neural Networks

In [4] Billsus and Pazzani identify limitations of the correlation-based approach (see equations 3.1 and 3.2) and present a novel idea to overcome these by representing the training data as a boolean matrix, which is then reduced in dimensionality for practical reasons to train a neural network for prediction.

The authors describe three problems which accompany correlation-based models in collaborative filtering:

- Correlation is based on items that both users have rated. However, overlap is typically small for large numbers of items, and thus correlation becomes not much realiable.

- As the model does not separate ratings into positive or negative classes, correlation can be close to zero, although there is predictive information available. Given user A's positive ratings are predictive for user B's negative ratings, but A's negative ratings do not suggest positive ratings for B. Then correlation might be considered small, even though there is useful information contained.

- Between two users, if there is not even one single item rated by both, there cannot be considered any correlation. But the fact that two users did not rate any common items so far does not necessarily imply, that their tastes do not correlate.

Regarding rating prediction as a classification task, the authors discuss the typical rating matrix (see table 3.1) to be learnt as column-wise examples for any algorithm to solve the classification problem.

Table 3.1: Rating matrix example.

	$\mathbf{I_1}$	$\mathbf{I_2}$	$\mathbf{I_3}$	$\mathbf{I_4}$	$\mathbf{I_5}$
$\mathbf{U_1}$	4		3		
$\mathbf{U_2}$		1		2	
$\mathbf{U_3}$	3	4	2		4
$\mathbf{U_4}$	4	2	1		**?**

For example, to predict the rating of user 4 for item 5, there could be provided three samples to a classification algorithm for user 4:

$$(\{4, \star, 3\} \rightarrow 4),\ (\{\star, 1, 4\} \rightarrow 2) \text{ and } (\{3, \star, 2\} \rightarrow 1).$$

Now in case the learning algorithm cannot deal with the missing values ($\star$), we already know *default voting*[6] as one option. However, the authors propose to tranform the rating matrix into an alternative boolean format, where for each combination of users, items and rating classes they note whether the combination has been observed or not. To reduce the complexity of the resulting matrix, the authors propose to discretize the [4]-ratings into classes *like* and *dislike*, such that the transformation of table 3.1 looks as follows (table 3.2):

Table 3.2: Rating matrix example transformed to boolean representation for user 4.

	$\mathbf{I_1}$	$\mathbf{I_2}$	$\mathbf{I_3}$
$\mathbf{U_1}$ **like**	1	0	1
$\mathbf{U_1}$ **dislike**	0	0	0
$\mathbf{U_2}$ **like**	0	0	0
$\mathbf{U_2}$ **dislike**	0	1	0
$\mathbf{U_3}$ **like**	1	1	0
$\mathbf{U_3}$ **dislike**	0	0	1
$\mathbf{U_4}$ **class**	like	dislike	dislike

Given this representation, the authors claim to be able to rely on virtually any supervised learning algorithm proposed in the literature. They also emphasize that hardly any researchers have attempted to solve the classification problem in the way of measuring the degree of correlation between features and class labels. Therefore Billsus and Pazzani propose to focus on algorithms, which more work out the discrimination of classes.

Since the tranformed matrix representation becomes too large to handle for practical datasets, the authors apply singular value decomposition (SVD) in order to determine the important *latent structure* of the training data. This is also motivated from the related document analysis task text classification, where two documents can very well handle the same topic though having only few words in common. Here Deerwester et al. propose latent semantic indexing (LSI), which is also based on SVD to reduce dimensionality.

Given the matrix A from table 3.2 with r rows and c columns, the SVD decomposes A into a product of three matrices U, Σ and V

$$A = U\Sigma V \tag{3.23}$$

with columns of U and V representing left and right singular vectors of A,

and Σ a diagonal matrix containing the corresponding singular values. U [V] is the $c \times c$ [$r \times r$] matrix containing singular vectors corresponding to the columns [rows] of A. The singular values of Σ provide information about the variance in the original data represented by the the singular vectors.

In order to reduce dimensionality low singular values and their corresponding singular vectors are set to zero. The remaining k left singular vectors scaled by their singular values are used as the new training data. New boolean item features to be classified are geometrically transformed into this new feature space by rotating the item feature v by U_k and scaling it by Σ_k

$$v_k = v^T U_k \Sigma_k^{-1} \tag{3.24}$$

which means that it is placed at the centroid of all the user ratings that it contains.

Given the transformed real-valued item features the authors train an artifical feedforward neural network for either solving rating prediction as a regression problem (using linear output neurons) or a classification problem (using logistic output units). For 20 users and up to 50 training samples the authors report as best an ANN with k input units, 2 hidden units and 1 output unit trained by backpropagation. The evaluation based on the F-measure shows that the proposed SVD/ANN method outperforms PPMCC as well as another ANN baseline with an alternative feature preprocessing step.

3.3 Clustering

We already looked at nearest neighborhood approaches and methods which try to solve rating prediction as a classification task. Next we will review two more techniques based on clustering in order to underline the diversity of strategies experienced in the past.

Collection Agents

The web page recommender system 'Fab'[2], designed by Balabanović and Shoham, pursues an hybrid approach:

- *Collection agents* act as collaborative filters and gather new web pages. These agents can each be regarded as a cluster of topics, aggregated from the most valuable interests of the users. In this way

collection agents keep a dynamic profile of terms to match and weight new pages against.

- The so collected pages are now passed to a central router, from where they are further distributed to the users via *selection agents*, each being aware of a single user's profile in order to apply content-based recommendation.

Due to this hybrid architecture the authors credit the Fab system to unite the advantages of both recommender system categories. In case there are few users the system reduces to content-based recommendation. New items can be handled based on content, as well. Users not matching any of the interest groups may receive a custom collection agent, such that other collection agents are not distorted — this of course, depends on the clustering and the amount of available resources for collection agents. Also one can adjust the influence of collaborative filtering by modifying the feature dimensionality of selection agents: when not considering any features within selection agents, the system reduces to collaborative filtering.

Principal Component Analysis

With *Eigentaste*[13] Goldberg et al. present a collaborative filtering model based on principal component analysis. Unlike settings where users may freely choose items to rate the authors here force users to rate a common *gauge set* of items, so that the resulting training data does not contain missing values. In contrast to the previously discussed SVD technique, from the dense training matrix there is directly computed the symmetric correlation matrix for users which is further linearly transformed by *Principal Component Analysis* (PCA).

First of all there is applied a *z-transformation* to each given rating $R_{u,i}$ such that

$$z_{ui} = \frac{R_{u,i} - \bar{R}_{\cdot,i}}{\sigma_i} \tag{3.25}$$

with

$$\bar{R}_{\cdot,i} = \frac{1}{|U|} \sum_{u \in U} R_{u,i} \tag{3.26}$$

the average rating per item i, and

$$\sigma_i^2 = \frac{1}{|U| - 1} \sum_{u \in U} (R_{u,i} - \bar{R}_{u,i})^2 \tag{3.27}$$

the sample variance of i.

Theorem 1. *Let Z be the matrix of z-transformed ratings. We receive Pearson's correlation matrix C among all users by computing*

$$C = \frac{1}{n-1} Z^T Z \tag{3.28}$$

Proof.

$$\begin{aligned}
c_{xy} &= \frac{1}{|U|-1} \sum z_x z_y \\
&= \frac{1}{|U|-1} \sum \frac{x_i - \bar{x}}{\sigma_x} \frac{y_i - \bar{y}}{\sigma_y} \\
&= \frac{\frac{1}{|U|-1} \sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sigma_x^2}\sqrt{\sigma_y^2}} \\
&= \frac{\frac{1}{|U|-1} \sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{|U|-1} \sum (x_i - \bar{x})^2}\sqrt{\frac{1}{|U|-1} \sum (y_i - \bar{y})^2}} \\
&= \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2}\sqrt{\sum (y_i - \bar{y})^2}}
\end{aligned}$$

□

The Eigendecomposition leads to a matrix of Eigenvalues Λ and matrices of Eigenvalues Σ and Σ^T

$$C = \Sigma \Lambda \Sigma^T \tag{3.29}$$

Given a linear transformation Y applied to Z with

$$Y = Z\Sigma^T \tag{3.30}$$

and the covariance matrix C_Y of Y equal to Λ allows sorting the resulting components $y \in Y$ according to highest Eigenvalues and filtering to p top components (called *principal components*), which explain the proportion of variance in the training data corresponding to Λ_p.

With $p = 2$ for Σ_p^T the authors project the training data R onto

$$x = R\Sigma_2^T \tag{3.31}$$

the Eigenplane in the vector space.

As a next step the projected data is clustered by a recursive rectangular clustering technique, but also other clustering methods could be used. As

the data used by the authors concentrates around the origin their approach is to decrease the size of clusters around this point. Starting from the minimum rectangle containing all data points rectangles are bisected recursively (when having the origin as a vertex) up to a given depth d. Algorithm 1 describes the process in detail. Figure 3.3 shows the examplary outcome for $d = 3$.

Algorithm 1: Recursively clustering projected data.

Input : projected set of coordinates P of form (x, y), depth $d \geq 0$
Output: $12d + 4$ clusters

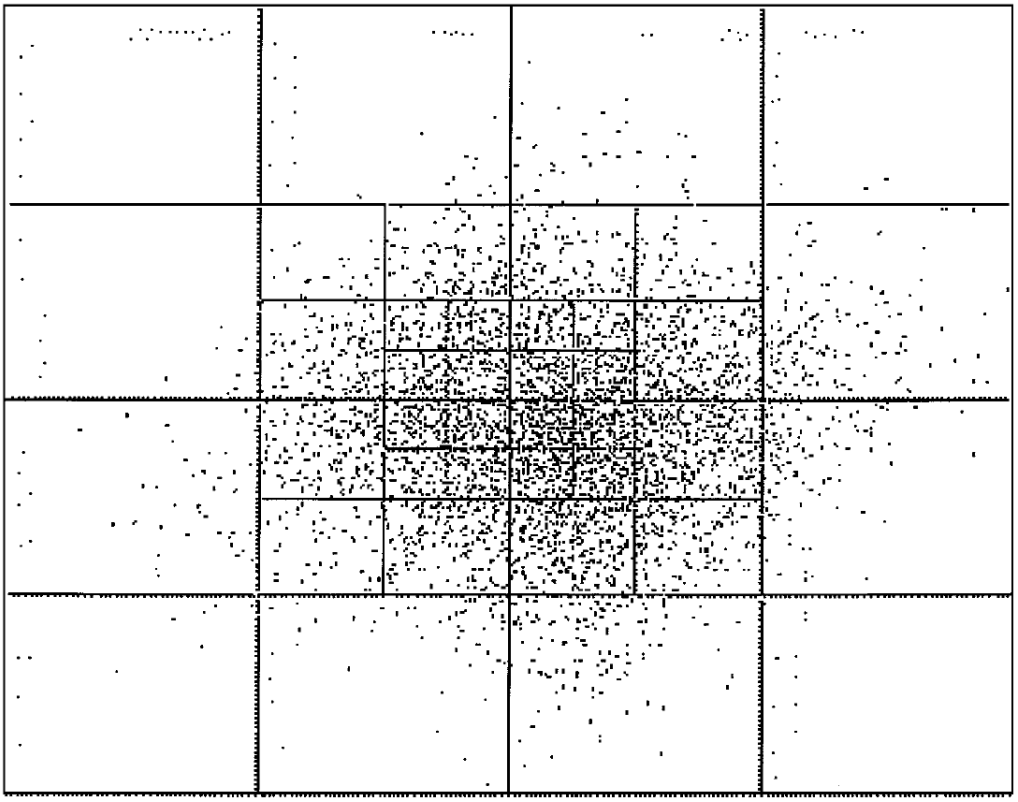
```
1  init rectangle R ← (x^P_min, y^P_min, x^P_max, y^P_max)
2  split R at (x_max − x_min)/2 [(y_max − y_min)/2] vertically [horizontally] to get 4 cells
   C.
3  if d > 0 then
4  |  foreach c ∈ C having origin as one of its vertices do
5  |  |  call routine recursively for data points in c and d − 1
6  |  |  C ← C\c
7  |  end
8  else
9  |  output each c remaining in C as a cluster
10 end
```

Now for each cluster there is predicted a rating for all items not being part of the gauge set so far, by averaging the ratings of users in the cluster that already rated the item. Decreasingly sorting these items by the predicted rating leads to a lookup table of recommendations for that cluster. Note that all the above steps from creating the correlation matrix up to computation of the recommendations per cluster can be precomputed offline.

For the online recommendation of items to a new user the system behaves as follows:

1. Collect ratings from the new user for all items of the gauge set.
2. Project this rating vector into the Eigenplane using the principal components.
3. Determine the cluster containing the projected vector.
4. Look up the corresponding recommendations, present them to the user and collect ratings.

Figure 3.3: Recursively clustering the data projected on Eigenplane ($p = 2$, $d = 3$).



The authors further compare their proposed method to a simple global mean predictor, as well as 1-Nearest-Neighbor and 80-Nearest-Neighbor algorithms based on PPMCC for weighting. The evaluation is based on the mean absolute error normalized to the rating scale $\tilde{R}$

$$NMAE = \frac{MAE}{\tilde{R}_{max} - \tilde{R}_{min}} \tag{3.32}$$

Results show that Eigentaste achieves equal qualitative performance as 80-NN and both outperform the remaining approaches. However, with online prediction runtime complexity in $O(k)$ compared to $O(|U|k)$ of 80-NN, Eigentaste is significantly faster (where k is the number of items in the gauge set).

3.4 Beyond the Box

To round off the retrospection we will furthermore review two contributions, the first sensibilizing to the human interface of recommender systems, which is an essential base for data collection, and second preprocessing steps which may improve accuracy and runtime to the benefit of applied algorithms.

Personal News Agent

The authors of [5] contribute a content-based recommender system additionally considering time-related user feedback designed for situations where access to a computer is restricted. Think of a user spending a lot of time in a car listening to the radio. Instead of switching through radio channels in order to receive information about topics of interest it might be more pleasant to be automatically provided with relevant stories.

To minimize the required bandwith for devices such as radios, the system shall transmit the news articles as textual information rather than audio. The news are then read to the user via speech synthesizer and also the user is enabled to interact by voice input.

Figure 3.4: Architecture of the Personal News Agent

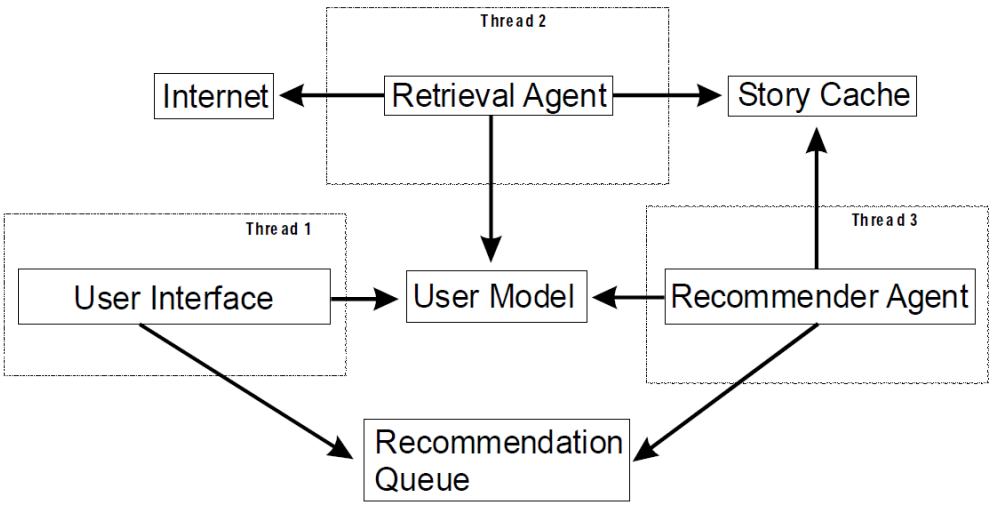


Figure 3.4 shows the general setup of the system consisting of three main components:

- The *Retrieval Agent* collects new articles from the internet based on the user model and stores them into a story cache.

- The *Recommender Agent* keeps an ordered queue of top rated news stories taken from the cache determined by the latest user model.
- The *User Interface* reads the top rated article from the queue and applies updates to the user model based on the user feedback.

In order to provide feedback the user may choose among the following options: *interesting, not interesting, I already know this, tell me more,* and *explain.*
The interface enables to measure the time how long a user has been listening to the story until feedback was provided, which the authors use to scale the provided rating to a continuous score as follows:

Story was rated as not interesting,	*score = 0.3 * p*
Story was rated as interesting,	*score = 0.7 + 0.3 * p*
User asked for more information,	*score = 1.0*

with p the proportion of a story the user has heard.

The demand for (1) representing a user's taste in different topics, (2) the ability to adapt to a user's changing preference even after a long training period and (3) the avoidance to present the same information twice motivates the authors to design a user model that keeps short- and long term interests separately and relies on the one or the other where approriate.

Requirement (2) and (3) are hosted on the *short-term model* which must be able to provide information about recently rated items as well as to identify stories already known by the user. Therefore the authors propose to use a nearest neighbor algorithm which corresponds almost exactly to the TF-IDF/cosine similarity method[22] we already looked at. However, they do not make use of the average prototype feature vector per class but keep a feature vector per single story. The weighted average score of all stories closer than a given threshold t_{min} then represents the predicted score for the new article, where the weight corresponds to the similarity.
If there exists at least one rated story closer than a threshold t_{max}, the new story is considered to be known and its predicted score is multiplied by a small constant in order to keep it at the end of the queue.
In cases where there is not identified even one near neighbor the article is passed to the long-term model.

The *long-term model* more devotes to requirement (1) and shall therefore model the general taste of the user. For this task the authors use a naïve Bayesian classifier based on $n \approx 200$ hand-selected domain-specific words which are used as features $f_1, ..., f_n$. News articles are then transformed into

feature vectors by checking the occurrence of those word-features. Assuming independence among features the probability of a story belonging to class c given its features is proportional to

$$P(c|f_1, ...f_n) \propto P(c) \prod_{i}^{n} P(f_i|c) \tag{3.33}$$

with application of Bayes' rule.
Additionally the authors require a story to have at least m features for which

$$P(f|c_{Interesting}) > P(f|c_{NotInteresting})$$

to allow classification as *interesting*, as well as

$$P(f|c_{NotInteresting}) > P(f|c_{Interesting})$$

to allow classification as *not interesting*.

To connect the short- and long-term model the authors propose to let in first place decide the short-term model whether it can handle a new story or not. If there is not enough information (no near neighbors) available yet, the long-term model gets its chance and may classify depending on the above restrictions. Algorithm 2 outlines the single prediction steps.

Algorithm 2: Combination of short- and long-term model for prediction in Personal News Agent.

Input : news story s, short-term model M_S, long-term model M_L
Output: predicted score p for the news story

```
if M_S can classify s then
    p = weighted average over nearest neighbors
    if s is too close to any neighbor then
        p = p * SMALL_CONSTANT
    end
else
    if M_L can classify s then
        p = probability estimated by naïve Bayes classifier
    else
        p = DEFAULT_SCORE
    end
end
```

Furthermore the system is able to *explain* the score of an article to the user in order to provide a certain degree of insight in terms of the article itself and the induced model. For this feature there are 4 predefined explanation templates available:

T1: *"This story received a **[high/low]** score, because you told me earlier that you were **[not]** interested in **[closest_headline]**."*

T2: *"I think you already know about this, because I told you earlier that **[closest_headline]**."*

T3: *"This story received a **[high/low]** score, because it contains the words f_1, ..., f_r."*

T4: *"The story received a default score, because it did not relate to any previously rated story, and did not contain enough informative words for classification."*

Given one of the explanations T1 to T3 the user may provide positive or negative feedback on the concept so that the model could better adapt to the user (even if taste changes over time) and achieve higher accurracy with less training data.
In case of T1 if the user provides negative feedback the closest story is taken out from the short-term model so that it won't affect new articles in the future. If feedback is positive the closest story is added once again to the short-term model to increase its weight. When an article is classified to be interesting but the user indicates that he heard of it before then t_{max} is decreased by a small constant.
If positive feedback is received on T2 nothing is changed. However, when the user provides negative feedback the threshold t_{max} is slightly increased.
For feedback on T3 there is constructed an artificial training example consisting of the words $f_1, ..., f_r$ that were most relevant for the classification, where influence is denoted as

$$i_f = log(\frac{P(f|c)}{P(f|not\ c)}) \tag{3.34}$$

The new training example is added to both, the short- and long-term model, with a class label corresponding to the feedback. By this for the long-term model simply the word frequencies are updated, which should improve classification. However, for the short-term model the new example is said to lead to a high similarity with future articles containing these few words,

so that the new articles will be ranked high/low in the queue depending on the class of the artificial example.

To evaluate Personal News Agent the authors had 10 users in touch with the system for up to 8 days and received averagely 300 ratings per user. Results show that the model converges to its best performance during the first 3 training days. The combination of short- and long-term model also achieves better results than each model taken individually. By measuring precision at the top 5 recommendations the authors show that time-coded score information outperforms boolean ratings due to an improved ordering of the queue. However, as time does not necessarily affect class membership accuracy and F_1-measure do not significantly change. Lastly the authors outline that conceptual feedback on explanations improve each accuracy and F_1-measure by 4.6%.

Instance Selection

Yu et al.[43] address the selection of training instances to improve accuracy and runtime of memory-based CF algrotihms as presented in equation 3.6 in combination with PPMCC. They propose four techniques of Training User Reduction for Collaborative Filtering (TURF1-TURF4) to yield a reduced set of training data per item $T_i' \subseteq T_i$ and predict based on T_i' instead of T_i.

The idea of TURF1 is to randomly choose an initial set T_i' of e.g. 150 users. Then for each remaining user $u \notin T_i'$ there is calculated the prediction for target item i based on the current T_i'. If prediction is wrong the user is added to T_i' since it seems to contain novel relevant information. Algrotihm 3 outlines the process in detail.

TURF2 tries to identify instances whose profiles are *stronger* and more *rational* than the profiles of other users.
Rationality of instance u for target item i is defined as

$$R_{u,i} = \sum_{j \in I_u \backslash i} I(R_{\cdot,i}; R_{\cdot,j}) \tag{3.35}$$

with $I_u \backslash i$ the set of items rated by u exluding i, and $I(R_{\cdot,i}; R_{\cdot,j})$ the *mutual information* between item i and j, so that rationality measures the sum of uncertainty about $\hat{R}_{u,i}$ reduced by u's profile.
Strength of rationality of instance u for target item i is defined as

$$S_{u,i} = \frac{1}{|I_u \backslash i|} R_{u,i} \tag{3.36}$$

the normalized rationality.

Algorithm 3: TURF1

Input : training data T with T_i the users who rated item i, initial size s
Output: reduced traninig data T'_i per item

```
1  foreach target item i in T do
2      if |T_i| > s then
3          initialize T'_i with s random users from T_i
4          foreach u ∈ T_i \ T'_i do
5              if u's rating on i is not correctly predicted by CF using T'_i
               then
6                  T'_i = T'_i ∪ T_u
7              end
8          end
9      end
10 end
```

Algorithm 4 denotes TURF2's process of selecting users with strong profiles. Again, after each set T'_i is determined, predictions are based on these reduced sets using the memory-based CF approach with PPMCC as weights.

TURF3 is a combination of TURF1 and TURF2 first identifying the top relevant profiles (TURF2) and then reducing to novel instances (TURF1). Due to the first reduction to relevant profiles the computational complexity and sensitivity to noise of TURF1 is decreased.

With TURF4 the authors propose a method to minimize the total number of users and storage consumption. Therefore they define the *utility* of user u as the number of target items N for which u serves as training user. Given a training set T TURF4 then eliminates the users with lowest utility such that 90% of the total utility remains in T'.

For the experimental results Yu et al. compare TURF1-TURF4 to the baseline (memory-based CF with PPMCC) without instance selection, as well as a random sampling approach where instances are selected according to a sampling rate. Table 3.3 shows the experimental results of the approaches for *All but one* protocol which means to learn all training instances and predict the one remaining item per user. For all methods b denotes the sampling rate. Random sampling should not become the method of choice for instance selection. It performs worse than the baseline and all TURF approaches in terms of accuracy. TURF1 and TURF2 are equal in

Algorithm 4: TURF2

Input : training data T with T_u the items rated by user u and T_i the users who rated item i, minimum set size to avoid oversampling a, sampling rate b

Output: reduced traninig data T_i' per item

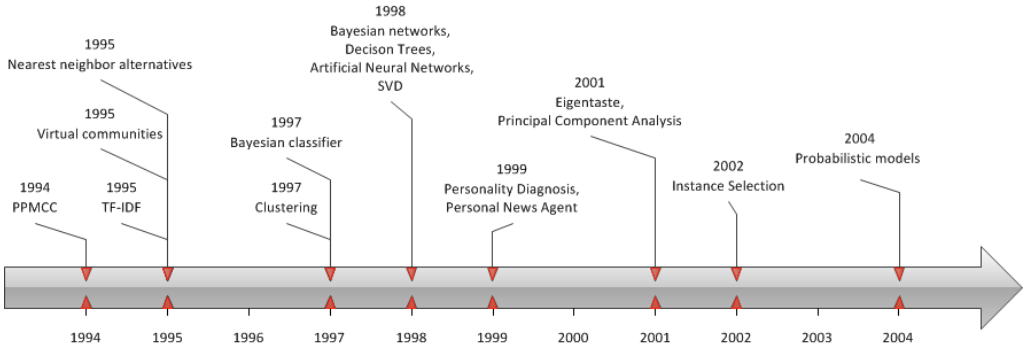
```
foreach user u in T do
    foreach item i in T_u do
        foreach other item j in T_u do
            compute I(R_{.,i}; R_{.,j})
        end
    end
end
foreach target item i in T do
    foreach user u in T_i do
        compute S_ui
    end
    T_i' ← max(a, |T_i| · b) strongest users
end
```

accuracy, however TURF2 eliminates more instances and so reduces runtime. TURF3, the combination of both, even improves accuracy and runtime. TURF4 may further decrease runtime in costs of prediction error and therefore represents a suitable gear to balance speed and quality. Results are also similar for *Given K* protocols where fewer ratings per user (e.g. 5, 10, 20) are considered.

Table 3.3: Experimental results of TURF1-TURF4 compared to baseline and random sampling for 'All but one' protocol.

	Runtime (ms)	**MAE**
Baseline	222	0.982
Random b**=0.125**	30	1.008
TURF1	122	0.959
TURF2 b**=0.125**	39	0.959
TURF3 b**=0.125**	30	0.947
TURF4 b**=0.125**	26	0.954
TURF4 b**=0.03125**	16	0.962

Figure 3.5: Looking back over different attempts to solve rating prediction. The figure provides a rough overview about the chronological order of discussed techniques.



3.5 Summary

In this chapter we learned about various techniques to solve the task of rating prediction by collaborative filtering and gained a first insight into the vast scope of approaches motivated from different areas, such as clustering user profiles, determining similarities among users or the creation of neural networks. One reshaped the rating matrix and applied dimesionality reduction while others focus on a subset of ratings to directly decompose the rating matrix.

Furthermore preprocessing steps (e.g. instance selection) can improve the quality and runtime of collaborative filtering algorithms. We also considered full system implementations where well-founded user interfaces, architectures and combinations of algorithms led to improved prediction accuracy.

Throughout this chapter we found similarities among the shown approaches, such as nearest neighbor techniques which are mostly related by applying different weights to the rating patterns. Also we learned about drawbacks of *online* techniques, and saw how e.g. model-based approaches like the PCA clustering method can shift the work to *offline* precomputation. Also we keep in mind that a probabilistic view of ratings, as presented by the classifying approaches, can lead to improved results.

While this chapter has more provided a categorical view about early recommender systems, figure 3.5 shall roughly outline a chronological order of the presented contributions. However, due to the great amount of publications (as shown in figure 1.1), figure 3.5 should by far not be understood as a complete representation of the past.

4 Matrix Factorization

In late 2006 Netflix, Inc. declared the *Netflix Prize Competition*[15][3] rewarding $1M for those who improve their existing movie recommender *Cinematch* by 10%. As a result the competition demonstrated that matrix factorization techniques outperform classic approaches, which we partially looked at in chapter 3. We will now have a look at the fundamentals of matrix factorization and its application to rating prediction, before introducing the most successful extension of the prize winning approach.

4.1 Principle of Matrix Factorization

The key idea of matrix factorization (as presented in [20]) is to approximate the observed, sparse rating matrix $R \in \mathbb{R}^{n \times m}$ by a dense matrix $\hat{R} \in \mathbb{R}^{n \times m}$, obtained from $\mathcal{R}(u, i)$ a linear combinations of *latent features* of user u and item i. Let n be the total number of users and m the total number of items in the system. Users and items are mapped to a k-dimensional latent feature space based on the observed ratings, such that each user u is represented by a vector $U_{u,\cdot} \in \mathbb{R}^k$, and each item i by $I_{i,\cdot} \in \mathbb{R}^k$, respectively, with $U \in \mathbb{R}^{n \times k}$ and $I \in \mathbb{R}^{m \times k}$. The dot product of $U_{u,\cdot}$ and $I_{i,\cdot}$ captures the interaction between u and i which approximates the rating of u for i:

$$R_{u,i} \approx \hat{R}_{u,i} = \mathcal{R}(u, i) = \sum_k U_{u,k} \cdot I_{i,k} \tag{4.1}$$

In other words $U_{u,\cdot}$ measures the amount of interest of a user in k item characteristics, individually pronounced per item by $I_{i,\cdot}$.

As R is sparse we cannot easily apply a low-rank matrix decomposition such as SVD to obtain predictive matrices U and I. Instead there are sought U and I such that $\hat{R}$ achieves the minimum squared error on each observation $(u, i) \in R$:

$$\min_{U,I} \sum_{(u,i) \in R} (R_{u,i} - \sum_k U_{u,k} \cdot I_{i,k})^2 \tag{4.2}$$

4.2 Model Training

In order to solve the optimization problem described by equation 4.2 there exist two popular algorithms, which we will have a look at.

(Stochastic) Gradient Descent

The gradient descent approach (related to [37]) finds a local minimum of the loss function $\mathcal{L}$ described in equation 4.2 by iteratively adjusting U and I in the opposite direction of its gradient $\nabla\mathcal{L}$, defined as

$$\nabla\mathcal{L} = \begin{pmatrix} \frac{\Delta\mathcal{L}}{\Delta U_{u,\cdot}} \\ \frac{\Delta\mathcal{L}}{\Delta I_{i,\cdot}} \end{pmatrix} \tag{4.3}$$

Let $\mathcal{P}(u,i)$ be the prediction error

$$\mathcal{P}(u,i) = R_{u,i} - \mathcal{R}(u,i) \tag{4.4}$$

Then for each training sample $(u,i) \in R$ in the observation, with the derivative of $\mathcal{L}$ with respect to $U_{u,\cdot}$

$$\frac{\Delta\mathcal{L}}{\Delta U_{u,\cdot}} = -2 \cdot \mathcal{P}(u,i) \cdot I_{i,\cdot} \tag{4.5}$$

as well as the derivative of $\mathcal{L}$ with respect to $I_{i,\cdot}$

$$\frac{\Delta\mathcal{L}}{\Delta I_{i,\cdot}} = -2 \cdot \mathcal{P}(u,i) \cdot U_{u,\cdot} \tag{4.6}$$

the algorithm collects update information for feature vectors of U and I — corresponding to their derivatives — from the observations, before adjusting them by a small proportion α of the update information in the opposite direction. Algorithm 5 shows the pseudocode of the full gradient learning process of U and I.

Note that choosing the learn rate α per iteration is not trivial and can itself be formulated as an optimization problem of its own. In practice, however, the learn rate is often chosen as a fixed value, considering that for a too low α the algorithm barely converges such that unreasonable numbers of iterations are necessary until convergence, whereas a too high α may leed to no convergence at all since the optimum is skipped back and forth in each iteration.

Algorithm 5: Learning an MF model by Gradient Descent.

Input : observed ratings R, latent dimensions k, iterations t, learn rate α

Output: latent feature matrices U and I

initialize U and I with small random values
while $t > 0$ **do**
  initialize U' and I' with 0
  foreach $(u, i) \in R$ **do**
    $U'_{u,\cdot} \leftarrow U'_{u,\cdot} + \frac{\Delta \mathcal{L}}{\Delta U_{u,\cdot}}$
    $I'_{i,\cdot} \leftarrow I'_{i,\cdot} + \frac{\Delta \mathcal{L}}{\Delta I_{i,\cdot}}$
  end
  for $u = 1 \rightarrow n$ **do**
    $U_{u,\cdot} \leftarrow U_{u,\cdot} - \alpha \cdot U'_{u,\cdot}$
  end
  for $i = 1 \rightarrow m$ **do**
    $I_{i,\cdot} \leftarrow I_{i,\cdot} - \alpha \cdot I'_{i,\cdot}$
  end
  $t \leftarrow t - 1$
end

The runtime complexity of GD is bound by

$$O(t \cdot ((k \cdot |R|) + (k \cdot n) + (k \cdot m))) = O(t \cdot k \cdot (|R| + n + m)) \tag{4.7}$$

and can be improved by a *stochastic* version (see [44]) which randomly picks tuples $(u, i) \in R$ and applies the update steps directly. Algorithm 6 outlines the process for our domain yielding a runtime complexity in

$$O(t \cdot k \cdot |R|) \tag{4.8}$$

For model implementations throughout the evaluation chapter we rely on standard gradient descent. Nonetheless we will briefly look at another training framework.

Alternating Least Squares

The idea of alternating least squares (ALS) (as presented in [45],[30]) is to reduce complexity of the loss function by presuming I as fixed for recomputation of U, and vice versa. These are the alternating steps, where in

Algorithm 6: Learning an MF model by Stochastic Gradient Descent.

Input : observed ratings R, latent dimensions k, iterations t, learn rate α
Output: latent feature matrices U and I

initialize U and I with small random values
while $t > 0$ **do**
  foreach $(u, i) \in R$ **do**
    $U'_{u,\cdot} \leftarrow \frac{\Delta \mathcal{L}}{\Delta U_{u,\cdot}}$
    $I'_{i,\cdot} \leftarrow \frac{\Delta \mathcal{L}}{\Delta I_{i,\cdot}}$
    $U_{u,\cdot} \leftarrow U_{u,\cdot} - \alpha \cdot U'_{u,\cdot}$
    $I_{i,\cdot} \leftarrow I_{i,\cdot} - \alpha \cdot I'_{i,\cdot}$
  end
  $t \leftarrow t - 1$
end

terms of recomputation of U (or I analogously) for each $U_{u,\cdot}$ the algorithm solves a separate least squares problem considering $\mathcal{I}^u_{i,\cdot}$ the feature vectors of items i rated by u, and $R^{I_u}_{u,\cdot}$ the observed rating vector of u restricted to rated item columns. We receive update operations when demanding the derivates of the so modified $\mathcal{L}'$ to be 0:

$$\frac{1}{2}\frac{\Delta \mathcal{L}'}{\Delta U_{u,\cdot}} = 0 \tag{4.9}$$

or

$$\frac{1}{2}\frac{\Delta \mathcal{L}'}{\Delta I_{i,\cdot}} = 0 \tag{4.10}$$

, respectively. We can transform as follows to yield the corresponding update operation for $U_{u,\cdot}$:

$$\frac{1}{2}\frac{\Delta\mathcal{L}'}{\Delta U_{u,\cdot}} = 0 \tag{4.11}$$

$$\Rightarrow \sum_{i\in I^u}(U_{u,\cdot}\cdot I_{i,\cdot} - R_{u,i})\cdot I_{i,\cdot} = 0$$

$$\Rightarrow \sum_{i\in I^u}(U_{u,\cdot}\cdot I_{i,\cdot})\cdot I_{i,\cdot} = \sum_{i\in I^u} R_{u,i}\cdot I_{i\cdot}$$

$$\Rightarrow \sum_{i\in I^u} I_{i,k}\cdot(U_{u,\cdot}\cdot I_{i,\cdot}) = \sum_{i\in I^u} I_{i,k}\cdot R_{u,i} \qquad \forall k$$

$$\Rightarrow \sum_{i\in I^u} I_{i,k}\cdot\sum_{k'} U_{u,k'}\cdot I_{i,k'} = \sum_{i\in I^u} I_{i,k}\cdot R_{u,i} \qquad \forall k$$

$$\Rightarrow \sum_{k'} U_{u,k'}\cdot\sum_{i\in I^u} I_{i,k}\cdot I_{i,k'} = \sum_{i\in I^u} I_{i,k}\cdot R_{u,i} \qquad \forall k$$

$$\Rightarrow \sum_{k'} U_{u,k'}\cdot(\mathcal{I}^u_{\cdot,k}\cdot\mathcal{I}^u_{\cdot,k'}) = \mathcal{I}^u_{\cdot,k}\cdot R^{I_u}_{u,\cdot} \qquad \forall k$$

$$\Rightarrow \sum_{k'} U_{u,k'}\cdot(\mathcal{I}^u\cdot\mathcal{I}^u)_{k,k'} = \mathcal{I}^u_{\cdot,k}\cdot R^{I_u}_{u,\cdot} \qquad \forall k$$

$$\Rightarrow U_{u,\cdot}\cdot(\mathcal{I}^u\cdot\mathcal{I}^u)_{k,\cdot} = \mathcal{I}^u_{\cdot,k}\cdot R^{I_u}_{u,\cdot} \qquad \forall k$$

$$\Rightarrow U_{u,\cdot}\cdot(\mathcal{I}^u\cdot\mathcal{I}^u) = \mathcal{I}^u\cdot R^{I_u}_{u,\cdot}$$

$$\Rightarrow U_{u,\cdot} = (\mathcal{I}^u\cdot\mathcal{I}^u)^{-1}\cdot(\mathcal{I}^u\cdot R^{I_u}_{u,\cdot})$$

Accordingly we obtain update operations for $I_{i,\cdot}$:

$$I_{i,\cdot} = (\mathcal{U}^i\cdot\mathcal{U}^i)^{-1}\cdot(\mathcal{U}^i\cdot R^{U_i}_{\cdot,i}) \tag{4.12}$$

with U_i the set of users who rated item i.

[45] contains further notions on parallelization of ALS, which makes it again favorable over gradient descent in case of parallel computing environments, since ALS is slower than GD in general. [30] outlines the computational complexity of *update operations* in ALS and yields

$$O(k^2\cdot|R| + n\cdot k^3) \tag{4.13}$$

for updating U, and

$$O(k^2 \cdot |R| + m \cdot k^3) \tag{4.14}$$

for updating I per single iteration, which we can summarize to a *total* ALS complexity upper bound

$$O(t \cdot (k^2 \cdot |R| + (n+m) \cdot k^3)) \tag{4.15}$$

4.3 Regularization

Training of the before mentioned model leads to overfitting as expressed by figure 4.1. While the error ϵ on the training data set (blue line) is continuously decreasing, there is a turning point (marked by the yellow warn sign) for the error on the test set (red line), where it starts increasing again. This is a consequence of the circumstance that the set of measured observations is typically biased from the real world, so that learning this data leads to biased models, as well. Now when making predictions about the real world with such a model, the bias leads to differing values and thus a higher error rate.

The phenomena of overfitting is typically addressed by *regularization*. In the case of matrix factorization there is added a regularization term ([42],[40],[32]) to the loss function, which penalizes large latent factor values:

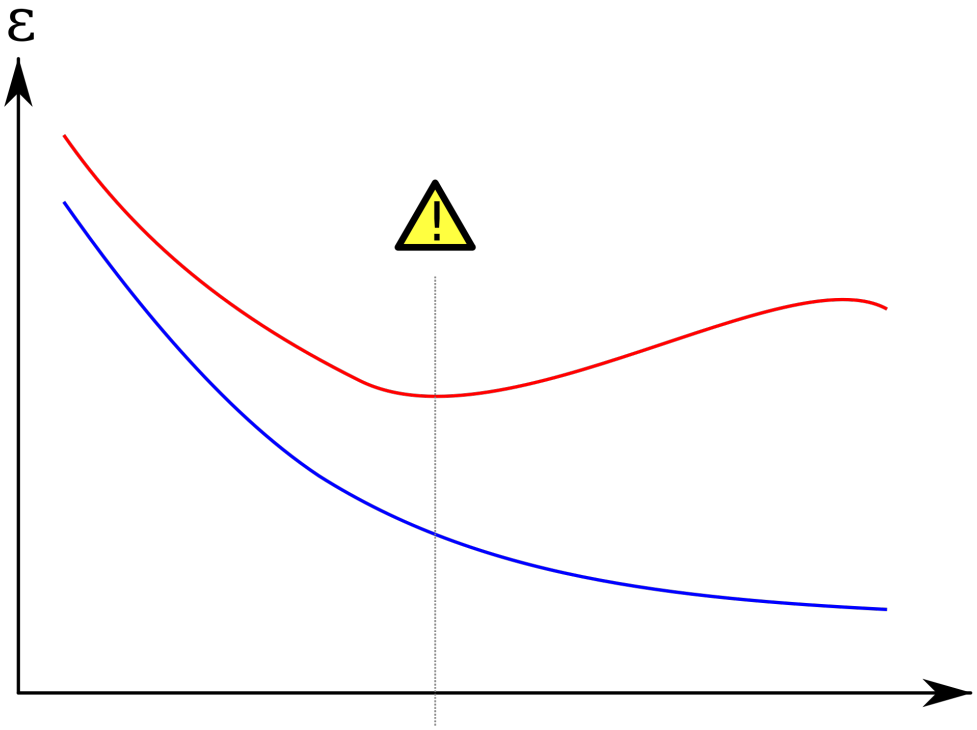
$$\mathcal{L}(R,U,I) = \sum_{(u,i)\in R} (R_{u,i} - \sum_k U_{u,k} \cdot I_{i,k})^2 + \frac{\lambda}{2}(||U||_F^2 + ||I||_F^2) \tag{4.16}$$

with

$$||A||_F = \sqrt{\sum_{i\in I} \sum_{j\in J} |A_{i,j}|^2} \tag{4.17}$$

the Frobenius norm.

Figure 4.1: Overfitting[1] — if the model adapts the bias from the 'real world' contained in the training data (blue line), the error ϵ on the test data (red line) increases again.



This regularization term postulates R-approximations where U and I contain small values to an extent determined by λ), so that the factorization can no longer overfit to the training data. This extension has become common practice to apply regularization and also resembles Tikhonov regularization/ridge regression, a well-known regularization technique for least squares problems in statistics. However, we will now look at matrix factorization from a probabilistic view, which allows to formalize a stronger foundation for regularization.

4.4 Probabilistic Matrix Factorization

As we have now a basic understanding of how matrix factorization works, how we can train a model and what we can use it for, we will next go into further details about the capabilities of the loss function. [38] motivate

[1] Image from `http://commons.wikimedia.org/wiki/File:Overfitting_svg.svg`

matrix factorization from a probabilistic view, which we will have a look at, as it will become our central framework for the incorporation of social trust information. Salakhutdinov et al. propose *probabilistic matrix factorization*, where they model ratings as a conditional distribution depending on U and I using the Gaussian normal distribution as probability density function which denotes as

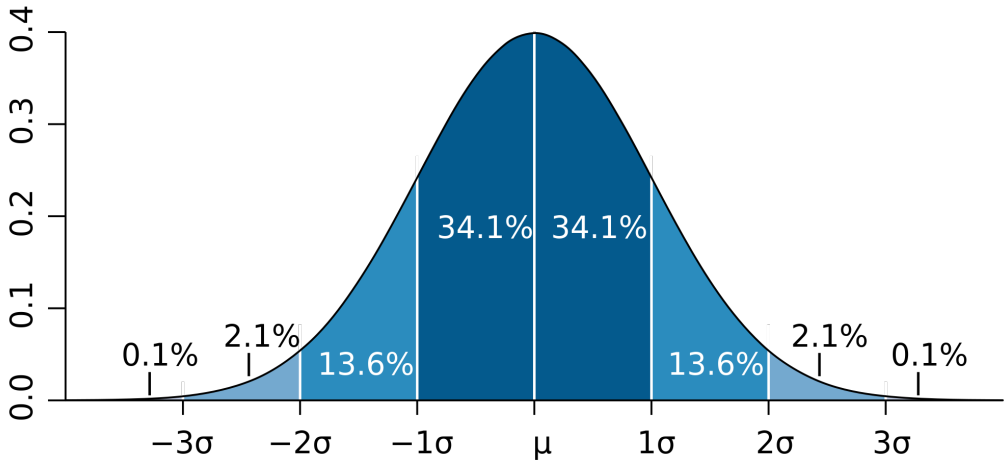
$$P(R|U, I, \sigma^2) = \prod_{u=1}^{n} \prod_{i=1}^{m} \mathcal{N}(R_{u,i}|U_{u,\cdot} \cdot I_{i,\cdot}, \sigma^2)^{X_{u,i}} \tag{4.18}$$

with mean $U_{u,\cdot} \cdot I_{i,\cdot}$, variance σ^2 and $X_{u,i}$ an indicator function defined as

$$X_{u,i} = \begin{cases} 1 & \text{, if } u \text{ rated } i \\ 0 & \text{, else.} \end{cases} \tag{4.19}$$

Figure 4.2 shows a graphical representation of the normal distribution which points out its typical data distribution along positive and negative intervals of the standard deviation σ.

Figure 4.2: Gaussian normal distribution[2] with mean μ and variance σ^2.



Furthermore there are drawn zero-mean Gaussian priors on user feature vectors

$$P(U|\sigma_U^2) = \prod_{u=1}^{n} \mathcal{N}(U_{u,\cdot}|0, \sigma_U^2 \mathbf{I}) \tag{4.20}$$

[2]Image from `http://commons.wikimedia.org/wiki/File:Standard_deviation_diagram.svg`

and item feature vectors

$$P(I|\sigma_I^2) = \prod_{i=1}^{m} \mathcal{N}(I_{i,\cdot}|0, \sigma_I^2 \mathbf{I}) \tag{4.21}$$

with $\mathbf{I}$ the identity matrix — which postulates feature values with mean 0 and variance σ_U^2 (or σ_I^2, respectively) before any data is observed.

Bayesian inference allows derivation of the posterior distribution $P(\theta|X, \alpha)$ from prior distribution $P(\theta|\alpha)$ and likelihood $P(X|\theta)$ by

$$P(\theta|X, \alpha) \propto P(X|\theta)P(\theta|\alpha) \tag{4.22}$$

with X the observed data, θ the parameters of the data distribution and α the hyperparameters.

Applied to our domain this yields

$$P(U, I|R, \sigma^2, \sigma_U^2, \sigma_I^2) \propto P(R|U, I, \sigma^2)P(U|\sigma_U^2)P(I|\sigma_I^2) \tag{4.23}$$

which we may solve according to the above defintions by

$$\begin{aligned} P(U, I|R, \sigma^2, \sigma_U^2, \sigma_I^2) = &\prod_{u=1}^{n}\prod_{i=1}^{m} \mathcal{N}(R_{u,i}|U_{u,\cdot} \cdot I_{i,\cdot}, \sigma^2)^{X_{u,i}} \\ &\prod_{u=1}^{n} \mathcal{N}(U_{u,\cdot}|0, \sigma_U^2 \mathbf{I}) \prod_{i=1}^{m} \mathcal{N}(I_{i,\cdot}|0, \sigma_I^2 \mathbf{I}) \end{aligned} \tag{4.24}$$

From the normal distribution with mean μ and variance σ^2

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.25}$$

we can rewrite equation 4.24 as

$$\begin{aligned} P(U, I|R, \sigma^2, \sigma_U^2, \sigma_I^2) = &\prod_{u=1}^{n}\prod_{i=1}^{m} \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(R_{u,i}-U_{u,\cdot}\cdot I_{i,\cdot})^2}{2\sigma^2}}\right)^{X_{u,i}} \\ &\prod_{u=1}^{n} \frac{1}{\sigma_U \mathbf{I}\sqrt{2\pi}} e^{-\frac{U_{u,\cdot}^2}{2\sigma_U^2 \mathbf{I}}} \prod_{i=1}^{m} \frac{1}{\sigma_I \mathbf{I}\sqrt{2\pi}} e^{-\frac{I_{i,\cdot}^2}{2\sigma_I^2 \mathbf{I}}} \end{aligned} \tag{4.26}$$

which yields

$$\begin{aligned}\ln P(U, I|R, \sigma^2, \sigma_U^2, \sigma_I^2) = &- \frac{1}{2\sigma^2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i}(R_{u,i} - U_{u,\cdot} \cdot I_{i,\cdot})^2 \\ &- \frac{1}{2\sigma_U^2} \sum_{u=1}^{n} U_{u,\cdot}^2 - \frac{1}{2\sigma_I^2} \sum_{i=1}^{m} I_{i,\cdot}^2 \\ &[-(\sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i}) \cdot \ln(\sigma\sqrt{2\pi}) \\ &- n \cdot k \cdot \ln(\sigma_U\sqrt{2\pi}) - m \cdot k \cdot \ln(\sigma_I\sqrt{2\pi})]\end{aligned} \tag{4.27}$$

after application of logarithm to the posterior distribution. Since the terms in brackets [...] do not depend on U or I they drop out for optimizing U and V.

Maximizing the log-posterior equals minimizing the negative, so that we finally receive the sum of squared errors objective function with quadratic regularization terms when hyperparameters are fixed:

$$\begin{aligned}\mathcal{L}(R, U, I) = &\frac{1}{2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,v}(R_{u,i} - U_{u,\cdot} \cdot I_{i,\cdot})^2 \\ &+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2\end{aligned} \tag{4.28}$$

with

$$\lambda_U = \frac{\sigma^2}{\sigma_U^2}, \quad \lambda_I = \frac{\sigma^2}{\sigma_I^2} \tag{4.29}$$

and $||\cdot||_F^2$ the Frobenius norm as denoted in eq. 4.17.

Additionally the authors propose to use a logistic function

$$g(x) = \frac{1}{1 + e^{-x}} \tag{4.30}$$

within the likelihood

$$P(R|U, I, \sigma^2) = \prod_{u=1}^{n} \prod_{i=1}^{m} \mathcal{N}(R_{u,i}|g(U_{u,\cdot} \cdot I_{i,\cdot}), \sigma^2)^{X_{u,i}} \tag{4.31}$$

to bound the range of predictions. Accordingly the observed ratings range $1, ..., V$ is mapped to the interval $[0, 1]$ by

$$t(x) = \frac{x - 1}{V - 1} \tag{4.32}$$

Again the model can be trained with one of the proposed methods (e.g. gradient descent).

4.5 Bias terms

A common extension to basic matrix factorization are bias terms ([20]). An option is to calculate the global average among all ratings and let U and I factorize the deviation from the average, instead of the rating itself. We can formulate this extension as

$$\mathcal{R}(u,i) = \omega^R + \sum_{k \in V_K} (U_{u,k} \cdot I_{i,k}) \tag{4.33}$$

where ω^R is the global average rating.

Furthermore one can introduce vectors which capture the per-user and per-item biases. Typically the rating averages for items vary from user to user, and also every item receives differently averaged ratings from the users. For example when regarding the rating matrix in table 2.1 the average rating of *Bob* is 4.5, while *Fred*'s is 3. For *High Noon* we observed a mean rating of 3.5 whereas *Titanic* is only rated 3.25 on average. For the difference between the global bias and per-user/-item bias there are added two vectors $\omega^U \in \mathbb{R}^n$ and $\omega^I \in \mathbb{R}^m$ to the prediction model:

$$\mathcal{R}(u,i) = \omega^R + \omega^U_u + \omega^I_i + \sum_{k \in V_K} (U_{u,k} \cdot I_{i,k}) \tag{4.34}$$

Regularization of the newly introduced bias vectors leads to the following loss function:

$$\mathcal{L}(R,U,I,\omega^R,\omega^U,\omega^I) = \frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m}(X_{u,i} \cdot \mathcal{P}(u,i)^2) + \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2 \tag{4.35}$$

$$+ \frac{\lambda_{\omega^U}}{2}||\omega^U||^2 + \frac{\lambda_{\omega^I}}{2}||\omega^I||^2$$

with

$$||v|| = \sqrt{\sum_{i \in I}(v_i)^2} \tag{4.36}$$

the Euclidean norm.

For model training with gradient descent we also need the derivates for the new bias terms, which denote as follows:

$$\frac{\Delta\mathcal{L}}{\Delta\omega^R} = -\sum_{u=1}^{n}\sum_{i=1}^{m}(X_{u,i}\cdot\mathcal{P}(u,i)) \tag{4.37}$$

$$\frac{\Delta\mathcal{L}}{\Delta\omega_u^U} = -\sum_{u=1}^{n}\sum_{i=1}^{m}(X_{u,i}\cdot\mathcal{P}(u,i)) + \lambda_{\omega^U}\cdot\omega_u^U \tag{4.38}$$

$$\frac{\Delta\mathcal{L}}{\Delta\omega_i^I} = -\sum_{u=1}^{n}\sum_{i=1}^{m}(X_{u,i}\cdot\mathcal{P}(u,i)) + \lambda_{\omega^I}\cdot\omega_i^I \tag{4.39}$$

Note that any other derivatives are not affected by these bias terms, so that they do not have to be updated here. Also keep in mind that plugging in bias terms is always possible for the matrix factorization framework, however, we do not make use of per-user/-item vectors until stated otherwise, as for the direct comparison of other extensions they *might* be regarded as a constant. We will investigate the impact of bias terms in the evaluation chapter. For any of the other following models we regard ω^R as contained implicitly. Since we are now equipped with a basic rating prediction framework we proceed to the central idea of the Netflix prize winning approach which extends the probabilistic framework by incorporating implicit rating information.

4.6 State of the Art: SVD++

In 2008 Yehuda Koren, who is also part of the Netflix prize winner team *BellKor's Pragmatic Chaos*, proposed probably one of the most boosting extensions to matrix factorization in recommender systems([21]). Among the list of his contributions to the winning team (outlined in [18]) *SVD++* already achieved an RMSE of 0.8814 with 200 factors after only 40 iterations, which is probably the best improvement of a single modification of matrix factorization, and therefore here referred to as the state of the art. There have also been made further improvements on SVD++ taking e.g. temporal information [19] or neighborhood [21] into account, however we will concentrate on the basic idea of the SVD++ model.

Instead of only considering the explicit rating value information SVD++ additionally complements each users features with another latent feature vector containing implicit information about which items where rated by the user, at all. To learn the implicit feature vectors the actual rating value

itself is ignored, so that only the binary fact *that* the item has been rated carries weight.

Formally the user features now add up to

$$U_{u,\cdot} + \frac{1}{\sqrt{|I_u|}} \sum_{j \in I_u} J_{j,\cdot} \tag{4.40}$$

so that the full prediction model denotes as

$$\mathcal{R}(u,i) = \sum_{k \in V_K} (I_{i,k} \cdot (U_{u,k} + \frac{1}{\sqrt{|I_u|}} \sum_{j \in I_u} J_{j,k})) \tag{4.41}$$

With the defintion of the prediction error (see eq. 4.4), we are again interested in the parameters which achieve the minimum quadratic error, thus

$$\mathcal{L}(R,U,I,J) = \frac{1}{2} \sum_{u=1}^{n} \sum_{i=1}^{m} (X_{u,i} \cdot \mathcal{P}(u,i)^2) + \frac{\lambda_U}{2} ||U||_F^2 + \frac{\lambda_I}{2} ||I||_F^2 + \frac{\lambda_J}{2} ||J||_F^2 \tag{4.42}$$

denotes the loss function with regularization terms.

Moreover, in order to perform gradient descent we need the derivates with respect to parameters $U_{u,k}$, $I_{i,k}$ and $J_{j,k}$:

$$\frac{\Delta \mathcal{L}}{\Delta U_{u,k}} = - \sum_{i=1}^{m} (X_{u,i} \cdot \mathcal{P}(u,i) \cdot I_{i,k}) + \lambda_U \cdot U_{u,k} \tag{4.43}$$

$$\frac{\Delta \mathcal{L}}{\Delta I_{i,k}} = - \sum_{u=1}^{n} (X_{u,i} \cdot \mathcal{P}(u,i) \cdot (U_{u,k} + \frac{1}{\sqrt{|I_u|}} \sum_{j \in I_u} J_{j,k})) + \lambda_I \cdot I_{i,k} \tag{4.44}$$

$$\frac{\Delta \mathcal{L}}{\Delta J_{j,k}} = - \sum_{u=1}^{n} \sum_{i=1}^{m} (X_{u,i} \cdot \mathcal{P}(u,i) \cdot X_{u,j} \cdot I_{i,k} \cdot \frac{1}{\sqrt{|I_u|}}) + \lambda_J \cdot J_{j,k} \tag{4.45}$$

5 Social-aware Matrix Factorization

With PMF we now have a suitable framework which allows us to incorporate social network information, e.g. in a similar way we already know from the SVD++ extension. In this chapter we will look at various models which are social-aware by considering the trust matrix (cf. table 2.3) in the factorization model.

5.1 SoRec

A first option to incorporate the social network information is proposed by Ma et al.[24]. Their model *SoRec* extends the probabilistic matrix factorization model by additionally drawing the observed trust matrix as conditional distribution of the form

$$P(T|U,V,\sigma_T^2) = \prod_{u=1}^{n}\prod_{v=1}^{n}\mathcal{N}(R_{u,i}|g(U_{u,\cdot}\cdot V_{v,\cdot}),\sigma_T^2)^{Y_{u,v}} \tag{5.1}$$

with mean $U_{u,\cdot}\cdot V_{v,\cdot}$, variance σ_T^2 and $Y_{u,v}$ an indicator function defined as

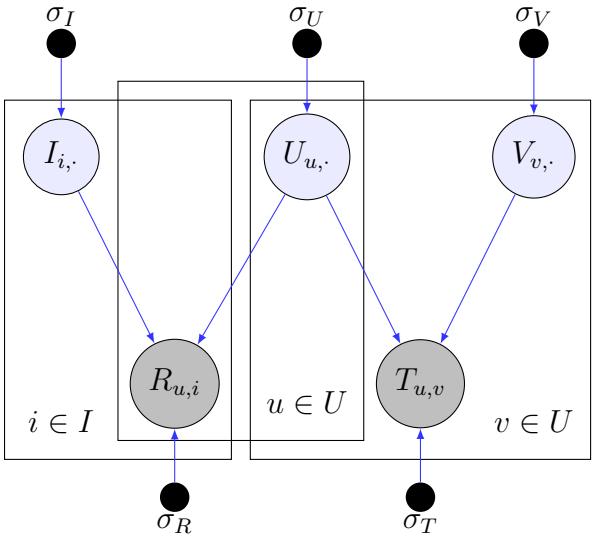
$$Y_{u,i} = \begin{cases} 1 & \text{, if } u \text{ trusts } v \\ 0 & \text{, else.} \end{cases} \tag{5.2}$$

In analogy to PMF there are assumed zero-mean Gaussian priors on the new feature vectors, as well:

$$P(V|\sigma_V^2) = \prod_{v=1}^{n}\mathcal{N}(V_{v,\cdot}|0,\sigma_V^2\mathbf{I}) \tag{5.3}$$

Figure 5.1 shows the *plate model* of the so extended PMF model. Rectangles can be interpeted as loops over the stated sets (e.g. $i \in I$), the blue

Figure 5.1: Plate model of SoRec. Blue nodes show feature vectors, black nodes result from the dot product of involved features. Black dots indicate prior assumptions and rectangles imply loops over the given set.



nodes represent feature vectors and black nodes result from the dot product of involved vectors. Also note the black dots which indicate the prior assumptions on their targets. Plate models are more like an informative view on the model, so one should not make statements about complexity based on them.

From the new model assumptions, again after application of Bayesian inference, we receive the following log posterior distribution:

$$\begin{aligned} \ln P(U, I, V | R, \sigma_R^2, \sigma_T^2, \sigma_U^2, \sigma_I^2, \sigma_V^2) = \\ - \frac{1}{2\sigma_R^2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i} (R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2 \\ - \frac{1}{2\sigma_T^2} \sum_{u=1}^{n} \sum_{v=1}^{n} Y_{u,v} (T_{u,v} - g(U_{u,\cdot} \cdot V_{v,\cdot}))^2 \\ - \frac{1}{2\sigma_U^2} \sum_{u=1}^{n} U_{u,\cdot}^2 - \frac{1}{2\sigma_I^2} \sum_{i=1}^{m} I_{i,\cdot}^2 - \frac{1}{2\sigma_V^2} \sum_{v=1}^{n} V_{v,\cdot}^2 \\ - [...] \end{aligned} \tag{5.4}$$

where maximizing this probability is equal to minimizing its negative in

case of fixed hyperparameters, and so in analogy to PMF we receive a sum of squared errors objective function with quadratic regularization terms:

$$\mathcal{L}(R,T,U,I,V) = \tag{5.5}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2$$

$$+ \frac{\lambda_T}{2}\sum_{u=1}^{n}\sum_{v=1}^{n} Y_{u,v}(T_{u,v} - g(U_{u,\cdot} \cdot V_{v,\cdot}))^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2 + \frac{\lambda_V}{2}||V||_F^2$$

with

$$\lambda_T = \frac{\sigma_R^2}{\sigma_T^2}, \quad \lambda_U = \frac{\sigma_R^2}{\sigma_U^2}, \quad \lambda_I = \frac{\sigma_R^2}{\sigma_I^2}, \quad \lambda_V = \frac{\sigma_R^2}{\sigma_V^2} \tag{5.6}$$

and $||\cdot||_F^2$ the Frobenius norm as denoted in eq. 4.17.

In order to learn the model with gradient descent we need the corresponding derivates as follows

$$\frac{\Delta\mathcal{L}}{\Delta U_{u,k}} = -\sum_{i=1}^{m}(X_{u,i} \cdot \mathcal{P}(u,i) \cdot I_{i,k}) \tag{5.7}$$

$$- \lambda_T \sum_{v=1}^{n}(Y_{u,v} \cdot \mathcal{Q}(u,v) \cdot V_{v,k}) + \lambda_U \cdot U_{u,k}$$

$$\frac{\Delta\mathcal{L}}{\Delta I_{i,k}} = -\sum_{u=1}^{n}(X_{u,i} \cdot \mathcal{P}(u,i) \cdot U_{u,k}) + \lambda_I \cdot I_{i,k} \tag{5.8}$$

$$\frac{\Delta\mathcal{L}}{\Delta V_{v,k}} = -\lambda_T \sum_{u=1}^{n}(Y_{u,v} \cdot \mathcal{Q}(u,v) \cdot U_{u,k}) + \lambda_V \cdot V_{v,k} \tag{5.9}$$

$$\tag{5.10}$$

, where

$$\mathcal{P}(u,i) = g'(U_{u,\cdot} \cdot I_{i,\cdot})(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot})) \tag{5.11}$$

and

$$\mathcal{Q}(u,v) = g'(U_{u,\cdot} \cdot V_{v,\cdot})(T_{u,v} - g(U_{u,\cdot} \cdot V_{v,\cdot})) \tag{5.12}$$

here also involve the logistic function described by eq. 4.30, with

$$g'(x) = \frac{e^x}{(1+e^x)^2} \tag{5.13}$$

5.2 Recommending with Social Trust Ensemble

A further way of considering trust in a matrix factorization model is presented by the same authors in [23]. Here the rating matrix is approximated as the weighted linear combination of the already seen user/item features

$$\alpha \cdot U_{i,\cdot} \cdot I_{i\cdot}$$

plus a factorization into socially connected user and item features

$$\frac{1-\alpha}{|N(u)|} \sum_{v \in N(u)} U_{v,\cdot} \cdot I_{i,\cdot}$$

, where $\alpha \in [0,1]$ allows to adjust the social influence and

$$N(u) = \{v \in U | T_{u,v} > 0\} \tag{5.14}$$

denotes the set of all users who are trusted by u.

The conditional distribution for observations thereby extends to

$$P(R|T,U,V,\sigma^2) = \tag{5.15}$$

$$\prod_{u=1}^{n} \prod_{i=1}^{m} \mathcal{N}(R_{u,i} | g(\alpha \cdot U_{u,\cdot} \cdot I_{i,\cdot} + \frac{1-\alpha}{|N(u)|} \sum_{v \in N(u)} U_{v,\cdot} \cdot I_{i,\cdot}), \sigma^2)^{X_{u,i}}$$

Again there are drawn zero-mean Gaussian priors on U and V, such that the log posterior distribution for U and V can be obtained via Bayesian inference as

$$\ln P(U,I|R,T,\sigma^2,\sigma_U^2,\sigma_I^2) = \tag{5.16}$$

$$- \frac{1}{2\sigma^2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i} (R_{u,i} - g(\alpha \cdot U_{u,\cdot} \cdot I_{i,\cdot} + \frac{1-\alpha}{|N(u)|} \sum_{v \in N(u)} U_{v,\cdot} \cdot I_{i,\cdot}))^2$$

$$- \frac{1}{2\sigma_U^2} \sum_{u=1}^{n} U_{u,\cdot}^2 - \frac{1}{2\sigma_I^2} \sum_{i=1}^{m} I_{i,\cdot}^2$$

$$- [...]$$

To solve the optimization problem we can equivalently calculate the minimum of the negative when keeping the hyperparameters fixed, so that we finally receive the following sum-of-squares loss function with regularization terms for the RSTE model:

$$\mathcal{L}(R,T,U,I) = \tag{5.17}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(\alpha \cdot U_{u,\cdot} \cdot I_{i,\cdot} + \frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)} U_{v,\cdot} \cdot I_{i,\cdot}))^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2$$

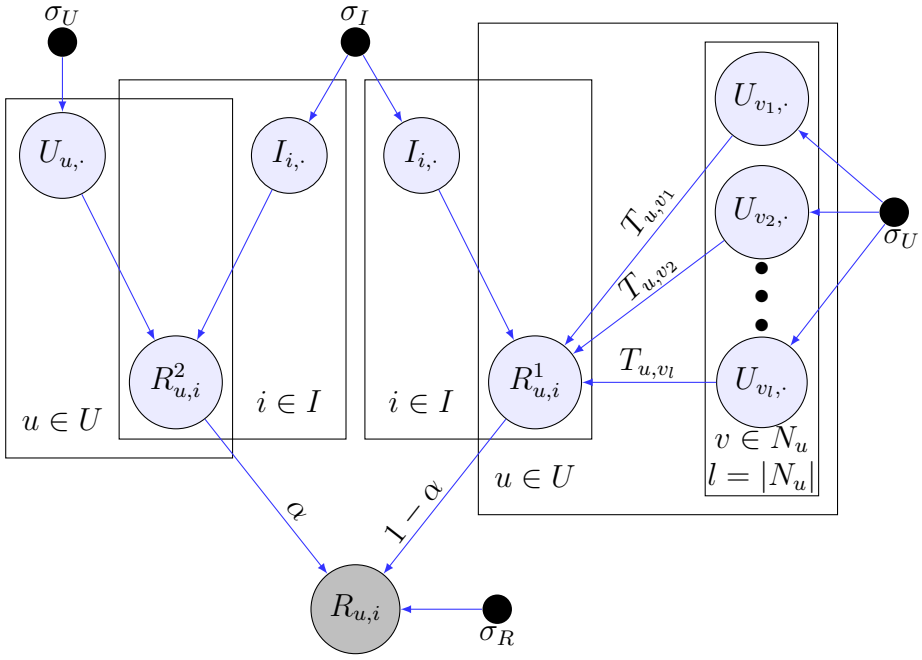
again with

$$\lambda_U = \frac{\sigma_R^2}{\sigma_U^2}, \quad \lambda_I = \frac{\sigma_R^2}{\sigma_I^2} \tag{5.18}$$

and $||\cdot||_F^2$ the Frobenius norm as denoted in eq. 4.17.

Figure 5.2 shows the plate model of the presented factorization approach. However, details and differences to SoRec will be discussed throughout the next chapter.

Figure 5.2: Plate model of RSTE.



In order to perform the local minimum search on the loss function with gradient descent the following derivates are necessary:

$$\begin{aligned}\frac{\Delta\mathcal{L}}{\Delta U_{u,k}} =& -\alpha\sum_{i=1}^{m}(X_{u,i}\cdot\mathcal{P}(u,i)\cdot I_{i,k}) \\ & -\sum_{v\in\mathcal{B}(u)}\frac{1-\alpha}{|N(v)|}\sum_{i=1}^{m}(X_{v,i}\cdot\mathcal{P}(v,i)\cdot I_{i,k}) \\ & +\lambda_U\cdot U_{u,k}\end{aligned} \tag{5.19}$$

$$\begin{aligned}\frac{\Delta\mathcal{L}}{\Delta I_{i,k}} =& -\sum_{u=1}^{n}(X_{u,i}\cdot\mathcal{P}(u,i)\cdot(\alpha\cdot U_{u,k}+\frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)}U_{v,k})) \\ & +\lambda_I\cdot I_{i,k}\end{aligned} \tag{5.20}$$

, where $\mathcal{B}(u)$ is the set of all users who trust u and

$$\begin{aligned}\mathcal{P}(u,i) =& g'(\alpha\cdot U_{u,\cdot}\cdot I_{i,\cdot}+\frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)}U_{v,\cdot}\cdot I_{i,\cdot}) \\ & \cdot(R_{u,i}-g(\alpha\cdot U_{u,\cdot}\cdot I_{i,\cdot}+\frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)}U_{v,\cdot}\cdot I_{i,\cdot}))\end{aligned} \tag{5.21}$$

describes the adapted logistic function introduced in eq. 4.30 and 5.13.

5.3 Social Matrix Factorization

The last social-aware factorization model we will look at is proposed by Jamali and Ester in [16]. The authors studied the STE model and miss that feature vectors of direct neighbors do not directly affect the active user's features, but only the predicted rating. Note that the plate model of RSTE given in here (fig. 5.2) and [16] differs from the original graphical representation in [23] in order to highlight this claim.

Jamali and Ester therefore model other related users' features to directly influence the user's features by approximating $U_{u,\cdot}$ to the sum of related vectors $U_{v,\cdot}$, where $v\in N(u)$:

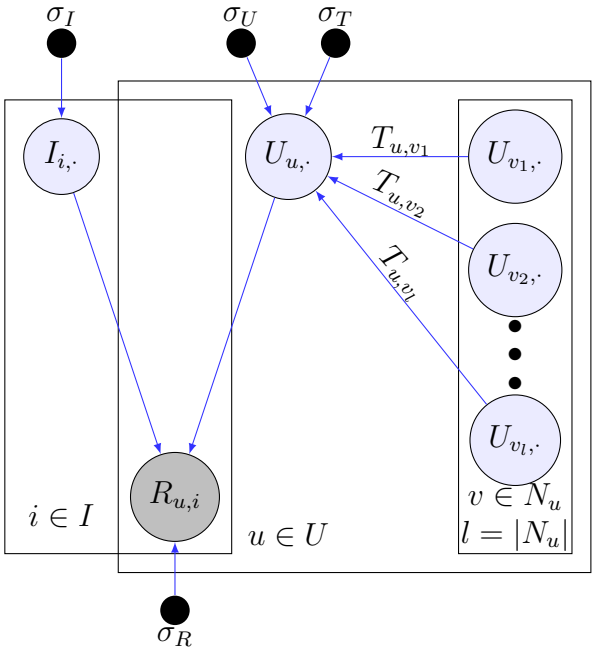
$$U_{u,\cdot}\sim\frac{1}{|N(u)|}\sum_{v\in N(u)}U_{v,\cdot} \tag{5.22}$$

The conditional probability $P(R|U, I, \sigma^2)$ and prior information for $P(I|\sigma_I^2)$ do not differ from PMF. However, the prior information of U is extended by the above assumption on social influence, so that the prior is formulated as the product of both Gaussian distributions:

$$P(U|T, \sigma_U^2, \sigma_T^2) = \prod_{u=1}^{n} \mathcal{N}(U_{u,\cdot}|0, \sigma_U^2 \mathbf{I}) \cdot \prod_{u=1}^{n} \mathcal{N}(U_{u,\cdot}|\frac{1}{|N(u)|} \sum_{v \in N(u)} U_{v,\cdot}, \sigma_T^2 \mathbf{I}) \tag{5.23}$$

Figure 5.3 further outlines this direct influence of other user features on the active user's feature indicated by the social relations (e.g. $T_{u,v}$) for the approximation of $R_{u,i}$ within the SocialMF model.

Figure 5.3: Plate model of SocialMF.



As before through Bayesian inference we receive the log posterior for

user and item features additionally depending on trust by

$$\begin{aligned} \ln P(U, I|R, T, \sigma_R^2, \sigma_T^2, \sigma_U^2, \sigma_I^2) = \\ - \frac{1}{2\sigma_R^2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2 \\ - \frac{1}{2\sigma_T^2} \sum_{u=1}^{n} (U_{u,\cdot} - \frac{1}{|N(u)|} \sum_{v \in N(u)} U_{v,\cdot})^2 \\ - \frac{1}{2\sigma_U^2} \sum_{u=1}^{n} U_{u,\cdot}^2 - \frac{1}{2\sigma_I^2} \sum_{i=1}^{m} I_{i,\cdot}^2 \\ - [...] \end{aligned} \tag{5.24}$$

We finally receive the SocialMF loss function with regularization terms by instead minimizing the negative log posterior under fixed hyperparameters, which denotes as

$$\begin{aligned} \mathcal{L}(R, T, U, I) = \\ \frac{1}{2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2 \\ + \frac{\lambda_T}{2} \sum_{u=1}^{n} (U_{u,\cdot} - \frac{1}{|N(u)|} \sum_{v \in N(u)} U_{v,\cdot})^2 \\ + \frac{\lambda_U}{2} ||U||_F^2 + \frac{\lambda_I}{2} ||I||_F^2 \end{aligned} \tag{5.25}$$

where

$$\lambda_U = \frac{\sigma_R^2}{\sigma_U^2}, \quad \lambda_I = \frac{\sigma_R^2}{\sigma_I^2}, \quad \lambda_T = \frac{\sigma_R^2}{\sigma_T^2} \tag{5.26}$$

and $|| \cdot ||_F^2$ the Frobenius norm as denoted in eq. 4.17.

For model training with gradient descent the following derivatives are

needed in the update steps:

$$\begin{aligned}\frac{\Delta\mathcal{L}}{\Delta U_{u,k}} = &-\sum_{i=1}^{m}(X_{u,i}\cdot\mathcal{P}(u,i)\cdot I_{i,k}) \\ &-\lambda_T(U_{u,k}-\frac{1}{|N(u)|}\sum_{v\in N(u)}U_{v,k}) \\ &+\lambda_T\sum_{v\in\mathcal{B}(u)}\frac{1}{|N(v)|}\cdot(U_{v,k}-\frac{1}{|N(v)|}\sum_{w\in N(v)}U_{w,k}) \\ &+\lambda_U\cdot U_{u,k}\end{aligned} \tag{5.27}$$

$$\frac{\Delta\mathcal{L}}{\Delta I_{i,k}} = -\sum_{u=1}^{n}(X_{u,i}\cdot\mathcal{P}(u,i)\cdot U_{u,k})+\lambda_I\cdot I_{i,k} \tag{5.28}$$

, where $\mathcal{B}(u)$ again describes the set of all users who trust u and

$$\mathcal{P}(u,i) = g'(U_{u,\cdot}\cdot I_{i,\cdot})\cdot(R_{u,i}-g(U_{u,\cdot}\cdot I_{i,\cdot}))$$

denotes the adapted logistic function introduced in eq. 4.30 and 5.13.

6 Theoretical Discussion

Throughout the following chapter we compare the proposed social-aware factorization models, as well as the SVD++ model to each other. First we work out similarities and differences of the model formulations and second derive a generic social-aware model. Lastly we compare the runtime complexities and outline how to speed up implementations by using an appropriate caching strategy.

6.1 Comparison of Factorization Models

In order to reduce complexity of formulas for a short comparison we introduce an informal notation for quadratic losses and regularization terms.

Let

$$< U_u, I_i >^R \quad :\approx \quad \frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2 \tag{6.1}$$

explain the prediction model within quadratic loss over all observed users and items, i.e. for PMF it informally substitutes the whole term on the right. The notation can be understood as *"R is decomposed into vectors of type U_u and I_i"*.

Furthermore let

$$|U_u|_{\lambda_U} \quad := \quad \frac{\lambda_U}{2}||U||_F^2 \tag{6.2}$$

substitute the regularization of matrix U as given by the right term.

Then for example with the above definitions, the PMF loss function (eq. 4.28) can informally be rewritten as

$$\mathcal{L}(R, U, I) =< U_u, I_i >^R + |U_u|_{\lambda_U} + |I_i|_{\lambda_I} \tag{6.3}$$

Table 6.1 summarizes the loss functions for the so far presented factorization models. While the basic PMF model simply captures the interaction

of users and items, SVD++ additionally involves the fact *that* a user has rated specific items, no matter how. The loss function of SVD++ therefore depends on the further parameter J, which allows to characterize the users interest in item i given the set of rated items. In other words J additionally captures the interaction of rated items and the item to predict.

Table 6.1: Loss functions of the presented models in simplified notation.

Model	**Loss Function**	
PMF	$< U_u, I_i >^R$	$+\lvert U_u\rvert_{\lambda_U} + \lvert I_i\rvert_{\lambda_I}$
SVD++	$< U_u + \frac{1}{\sqrt{\lvert I_u\rvert}} \sum_{j \in I_u} J_{j,\cdot}, I_i >^R + \lvert J_j\rvert_{\lambda_J}$	$+\lvert U_u\rvert_{\lambda_U} + \lvert I_i\rvert_{\lambda_I}$
SoRec	$< U_u, I_i >^R + < U_u, V_v >^T_{\lambda_T} + \lvert V_v\rvert_{\lambda_V}$	$+\lvert U_u\rvert_{\lambda_U} + \lvert I_i\rvert_{\lambda_I}$
RSTE	$< \alpha U_u + \frac{1-\alpha}{\lvert N(u)\rvert} \sum_{v \in N(u)} U_v, I_i >^R$	$+\lvert U_u\rvert_{\lambda_U} + \lvert I_i\rvert_{\lambda_I}$
SocialMF	$< U_u, I_i >^R + \lvert U_u - \frac{1}{\lvert N(u)\rvert} \sum_{v \in N(u)} U_v\rvert_{\lambda_T}$	$+\lvert U_u\rvert_{\lambda_U} + \lvert I_i\rvert_{\lambda_I}$

Next we have SoRec, which also introduces a further parameter V, however, here to factorize the trust matrix T. The parameter V is not involved for the prediction of R itself, but it acts as mate of U for the decomposition of T. Thus U participates in both decompositions at the same time: the decomposition of R and the decomposition of T. This could also be interpreted as a regularization of U, as the information in U is concurrently restricted to a degree of well performance on the factorization of T.

In the RSTE model the predicted rating of item i for user u is influenced by predictions of i for friends, where friends are defined by $N(u)$ the set of users who have a social relation to user u. The extent of this influence is controlled via an additional hyperparameter α, depending on the data set. For example, setting $\alpha = 1$ completely ignores ratings of friends, thus reducing the RSTE model to pure PMF. Otherwise $\alpha = 0$ makes the prediction completely based on friends' ratings, so that the user latent features do not have any impact, at all.

Furthermore there has been presented SocialMF, which penalizes difference among user and friend latent features via regularization, thus making user and friend latent features directly dependent on each user. Here as well the influence of friends can be controlled via an additional hyperparamater λ_T.

This suggests, though looking quite different at a first glance, RSTE and SocialMF are not that dissimilar in what they actually pursuit. We will

therefore step into the details of both RSTE and SocialMF formulations in order to work out the similarities explicitly.

6.2 Alignment of RSTE and SocialMF

Thoughout this section we will work out the difference of RSTE and SocialMF more explicitly by first transposing the prediction model of RSTE into the SocialMF prediction model and second by bringing the parameter assumptions of both loss functions closer together.

Alignment of the Prediction Model

The loss function of RSTE has already been derived in section 5.2:

$$\mathcal{L}(R,T,U,I)^{RSTE} = \tag{6.4}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(\alpha \cdot U_{u,\cdot} \cdot I_{i,\cdot} + \frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)} U_{v,\cdot} \cdot I_{i,\cdot}))^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2$$

thus we define

$$\mathcal{R}(u,i|T,U,I)^{RSTE} := \alpha \cdot U_{u,\cdot} \cdot I_{i,\cdot} + \frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)} U_{v,\cdot} \cdot I_{i,\cdot} \tag{6.5}$$

where $|T,U,I$ of the prediction model $\mathcal{R}(u,i|T,U,I)^{RSTE}$ shall express the dependence of $\mathcal{R}(u,i)$ on T, U and I. Next we factor out $I_{i,\cdot}$ and receive

$$\mathcal{R}(u,i|T,U,I)^{RSTE} := (\alpha \cdot U_{u,\cdot} + \frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)} U_{v,\cdot}) \cdot I_{i,\cdot} \tag{6.6}$$

We further define

$$U^*_{u,\cdot} := \alpha \cdot U_{u,\cdot} + \frac{1-\alpha}{|N(u)|}\sum_{v\in N(u)} U_{v,\cdot} \tag{6.7}$$

and substitute

$$\mathcal{R}(u,i|T,U,I)^{RSTE} = U^*_{u,\cdot} \cdot I_{i,\cdot} \tag{6.8}$$

Now let

$$\mathcal{L}(R,T,U',I)^{SocialMF} = \tag{6.9}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U'_{u,\cdot} \cdot I_{i,\cdot}))^2$$

$$+ \frac{\lambda_T}{2}\sum_{u=1}^{n}(U'_{u,\cdot} - \frac{1}{|N(u)|}\sum_{v\in N(u)} U'_{v,\cdot})^2$$

$$+ \frac{\lambda_{U'}}{2}||U'||_F^2 + \frac{\lambda_I}{2}||I||_F^2$$

be the SocialMF loss function (cf. equation 5.25) derived in section 5.3, for which we analogously define the prediction term as

$$\mathcal{R}(u,i|U',I)^{SocialMF} := U'_{u,\cdot} \cdot I_{i,\cdot} \tag{6.10}$$

Given that the prediction terms of RSTE and SocialMF can be equalized as follows:

$$\mathcal{R}(u,i|T,U,I)^{RSTE} = U^*_{u,\cdot} \cdot I_{i,\cdot} = \mathcal{R}(u,i|U^*,I)^{SocialMF} \tag{6.11}$$

With equation 6.11 we substitue the prediction term of the RSTE loss function, which yields

$$\mathcal{L}(R,T,U,I)^{RSTE} = \tag{6.12}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(\mathcal{R}(u,i|U^*,I)^{SocialMF})^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2$$

so that we successfully expressed the RSTE prediction term as a SocialMF prediction term.

Alignment of Parameter Assumptions

To further reduce the contrast of both loss functions we will next try to bring the parameter assumptions more into line, since they are still different. On

the one hand, for RSTE we defined

$$U^*_{u,\cdot} := \alpha \cdot U_{u,\cdot} + \frac{1-\alpha}{|N(u)|} \sum_{v \in N(u)} U_{v,\cdot}$$

where

$$U_{u,\cdot} \sim \mathcal{N}(U_{u,\cdot}|0, \sigma_U^2 \mathbf{I}) \tag{6.13}$$

is drawn from a zero-mean Gaussian distribution. On the other hand the priors on U' in SocialMF (cf. equation 5.23) are defined as

$$U'_{u,\cdot} \sim \mathcal{N}(U'_{u,\cdot}|0, \sigma_U'^2 \mathbf{I}) \cdot \mathcal{N}(U'_{u,\cdot}|\frac{1}{|N(u)|} \sum_{v \in N(u)} U'_{v,\cdot}, \sigma_T^2 \mathbf{I}) \tag{6.14}$$

In order to further equalize parameter assumptions we rewrite our defintion of $U^*_{u,\cdot}$ from equation 6.7 as

$$U^*_{u,\cdot} = M \cdot U_{u,\cdot} \tag{6.15}$$

where

$$M = \alpha \cdot \mathbf{I} + \frac{1-\alpha}{|N(u)|} \cdot \mathbf{I} \cdot A \tag{6.16}$$

with $\mathbf{I}$ the identity matrix and A the binary adjacency matrix of the trust network. We may transpose equation 6.15 to

$$U_{u,\cdot} = M^{-1} \cdot U^*_{u,\cdot} \tag{6.17}$$

in case of M^{-1} is invertible, which we will prove in the following.

Definition 1. *A square matrix is* strictly diagonally dominant *if each diagonal element in absolute value is greater than the sum of the absolute values of the off-diagonal elements in that row. If the matrix in question is A and has generic elements $a_{i,j}$, the strict diagonal dominance is expressed by*

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}| < |a_{i,i}| \tag{6.18}$$

(definition from [8])

Theorem 2. *Every diagonally dominant matrix is* invertible.

The proof is given in [8].

Theorem 3. *M is strictly diagonally dominant for $\alpha > 0.5$.*

Proof. In equation 6.16 we have chosen M such that its diagonal entries are set to α and all adjacent entries of row u are set to $\frac{1-\alpha}{|N(u)|}$. Thus, since all values in M are greater or equal to zero, strict diagonal dominance is given if for all rows u in M

$$\sum_{v \in N(u)} \frac{1-\alpha}{|N(u)|} < \alpha$$

which simplifies to

$$1 - \alpha < \alpha$$

so that M is strictly diagonal dominant for

$$\alpha > 0.5$$

□

From theorem 3 follows M is invertible for larger values of α and equation 6.17 holds, so that we can substitue in the loss function of RSTE as follows:

$$\mathcal{L}(R, T, U, I)^{RSTE} = \tag{6.19}$$

$$\frac{1}{2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i} (R_{u,i} - g(\mathcal{R}(u, i|U^*, I)^{SocialMF})^2$$

$$+ \frac{\lambda_U}{2} ||M^{-1} \cdot U^*_{u,\cdot}||_F^2 + \frac{\lambda_I}{2} ||I||_F^2$$

which allows us to drop the now unbound parameter U in favor of directly optimizing U^*. Furthermore we rename

$$M^{RSTE} = M^{-1} \tag{6.20}$$

in order to formalize the final RSTE loss function as

$$\mathcal{L}(R, T, U^*, I)^{RSTE} = \tag{6.21}$$

$$\frac{1}{2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i} (R_{u,i} - g(\mathcal{R}(u, i|U^*, I)^{SocialMF})^2$$

$$+ \frac{\lambda_{U^*}}{2} ||M^{RSTE} \cdot U^*_{u,\cdot}||_F^2 + \frac{\lambda_I}{2} ||I||_F^2$$

Now for SocialMF we can also define the influence of friends as a matrix by

$$M^{SocialMF} = \mathbf{I} - \frac{1}{|N(u)|} \cdot \mathbf{I} \cdot A \tag{6.22}$$

Thus we may formalize the final SocialMF loss function as

$$\begin{aligned} \mathcal{L}(R, T, U', I)^{SocialMF} = & \\ & \frac{1}{2} \sum_{u=1}^{n} \sum_{i=1}^{m} X_{u,i} (R_{u,i} - g(\mathcal{R}(u, i|U', I)^{SocialMF}))^2 \\ & + \frac{\lambda_T}{2} ||M^{SocialMF} \cdot U'_{u,\cdot}||_F^2 + \frac{\lambda_{U'}}{2} ||U'||_F^2 \\ & + \frac{\lambda_I}{2} ||I||_F^2 \end{aligned} \tag{6.23}$$

On the difference of RSTE and SocialMF

Equations 6.21 and 6.23 suggest that the remaining difference between both model formulations is the choice of matrices M^{RSTE} versus $M^{SocialMF}$, as well as the additional regularization of U in SocialMF. Followingly both models are very similar in their computational foundation, however, the main difference of the models underlies the diverse definition of the *similarity matrix* M.

In order to make this point more clear, we will further generalize both loss functions to a generic social-aware matrix factorization model which is able to adapt to either RSTE or SocialMF depending on the selection of the similarity matrix.

6.3 Towards a Generic, Social-Aware Matrix Factorization Model

From equations 6.21 and 6.23 we derive a generic model, denoted as

$$\mathcal{L}(R,T,U,I)^{GEN} = \tag{6.24}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(\mathcal{R}(u,i)))^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2$$

$$+ \frac{\lambda_T}{2}||M \cdot U_{u,\cdot}||_F^2$$

with M a freely selectable matrix for the incorporation of trust, and $\mathcal{R}(u,i)$ the prediction model.

The generic model can express the RSTE loss function by

$$\mathcal{L}(R,T,U,I)^{GEN} - \frac{\lambda_U}{2}||U||_F^2 = \mathcal{L}(R,T,U,I)^{RSTE} \tag{6.25}$$

when choosing $\mathcal{R}(u,i) = U_{u,\cdot} \cdot I_{i,\cdot}$ and $M = M^{RSTE}$.

Furthermore the SocialMF loss function can be obtained from the generic loss function since

$$\mathcal{L}(R,T,U,I)^{GEN} = \mathcal{L}(R,T,U,I)^{SocialMF} \tag{6.26}$$

when also choosing $\mathcal{R}(u,i) = U_{u,\cdot} \cdot I_{i,\cdot}$ but here $M = M^{SocialMF}$.

The choice of the similarity matrix may depend on the real world data set, so that the application should decide on which similarity matrix to rely for more or less accurate social-aware prediction of ratings.

6.4 Notions on Complexity

We will now look at the different complexities of the proposed social-aware factorization models. Compared to [24], [23] and [16] we implement a slightly different stop criterion for gradient descent as presented in algorithm 5, where we do not run until convergence of the loss function but stop after a fixed number of iterations t. With a k-fold cross validation evaluation protocol we determine t by iterating until the best sum of RMSEs for all ratings in the validation set is reached. Please refer to chapter

8 for details of the evaluation protocol. Now since we do not run until convergence, we will present complexities of the prediction terms $\mathcal{R}(u,i)$ of the models instead of the full loss functions. Furthermore we will compare the complexities of the gradients.

The RSTE model, as given in section 5.2, does not directly modify the user latent features, so that for each prediction the connected users of the active user have to be concerned. Therefore the complexity of the RSTE prediction term $\mathcal{R}(u,i)$ is

$$\begin{aligned} & O(k + |T| \cdot k) \\ = \; & O(k \cdot (1 + |T|)) \\ = \; & O(k \cdot |T|) \end{aligned} \tag{6.27}$$

Note that we do generally not consider implementation optimizations here, such as e.g. hash maps in order to cache intermediate results, but we will regard the pure mathematical formulations of the original models. Therefore we also do *not* rewrite $|\mathrm{T}|$ in the above equation by e.g. $\overline{|T^u|}$ as the average number of friends of u.

Next to the prediction term, for gradient descent also the gradient complexities are interesting to investigate. We make a further simplification in that we do not consider the derivatives of full feature vectors but only regard derivatives for a specific latent feature k. Thus the complexity for updating $U_{u,k}$ in RSTE is bound by

$$\begin{aligned} & O(m \cdot (k + |T| \cdot k) + |T| \cdot m \cdot (k + |T| \cdot k)) \\ = \; & O(m \cdot ((k + |T| \cdot k) + |T| \cdot (k + |T| \cdot k))) \\ = \; & O(m \cdot (k \cdot (1 + |T|) + |T| \cdot k \cdot (1 + |T|))) \\ = \; & O(m \cdot k \cdot ((1 + |T|) + |T| \cdot (1 + |T|))) \\ = \; & O(m \cdot k \cdot (|T| + |T|^2)) \\ = \; & O(m \cdot k \cdot |T|^2) \end{aligned} \tag{6.28}$$

since we would have to iterate through all items m and calculate the quadratic loss $(k + |T| \cdot k)$, plus we need to do the same for each neighbor of the user: $|T| \cdot m \cdot (k + |T| \cdot k))$. The penalization effort of the regularization drops out as a constant in O-notation when considering the complexity of a single latent feature.

The complexitiy for updating an item latent feature $I_{i,k}$ in RSTE is bound by

$$\begin{aligned} & O(n \cdot (k + (|T| \cdot k))) \\ = & O(n \cdot k \cdot (1 + |T|)) \\ = & O(n \cdot k \cdot |T|) \end{aligned} \tag{6.29}$$

as we need to iterate through all users here and also have to calculate the quadratic loss. Here, for example, there is room to speed up the full gradient determination $U_{u,\cdot}$ and $I_{i,\cdot}$ by simply caching the computation of quadratic losses and sharing them where necessary. The same strategy holds for other models as well, so that it is generally advisable for social-aware models to cache the quadratic losses $\mathcal{P}(u,i)$ over the iteration t, no matter what the prediction term $\mathcal{R}(u,i)$ looks like. With this caching strategy the efforts of calculating $\mathcal{P}(u,i)$ reduces to a constant within gradient complexities.

In SocialMF, as presented in section 5.3, the user latent feature are modified within the compuation of gradients, so that the complexity of predicting $\mathcal{R}(u,i)$ reduces to

$$O(k) \tag{6.30}$$

which equals the prediction complexity of PMF.

Moreover the computational effort to determine $U_{u,k}$ is bound by

$$\begin{aligned} & O(m \cdot k + |T| + |T| \cdot |T|) \\ = & O(m \cdot k + |T| + |T|^2) \\ = & O(m \cdot k + |T|^2) \end{aligned} \tag{6.31}$$

which already suggests that iterations in SocialMF are performed faster than in RSTE.

Since there is no interaction with trust when computing the update for $I_{i,k}$ in SocialMF, the effort of this operation is simply bound by

$$O(n \cdot k) \tag{6.32}$$

so that there is again no difference to PMF.

In section 5.1 there has been presented SoRec, which also factorizes the trust matrix next to the rating matrix by introducing the further feature matrix V to interact with U. The prediction model itself is not affected by this approach, so that prediction complexity is again equal to PMF:

$$O(k) \tag{6.33}$$

The situation slightly changes for updating $U_{u,k}$, since here the quadratic loss for the trust matrix has to be determined, as well:

$$O(m \cdot k + n \cdot k) \tag{6.34}$$

$$=O(k \cdot (n + m))$$

Note that also here a caching strategy for the quadratic loss $\mathcal{Q}(u, v)$ of the trust matrix further improves this complexity.

The computational effort to determine $I_{i,k}$ is again bound by

$$O(n \cdot k) \tag{6.35}$$

which is again equal to PMF.

Furthermore the full SoRec gradient consists of the additional derivate of V, for which we also provide the update complexity per single feature $V_{v,k}$ as

$$O(n \cdot k) \tag{6.36}$$

since for all n users the quadratic loss $\mathcal{Q}(u, v)$ of the trust matrix has to be computed.

Table 6.2 summarizes the given complexities of predictions and update operations of the proposed social-aware factorization models.

Table 6.2: Runtime complexities of RSTE, SocialMF and SoRec.

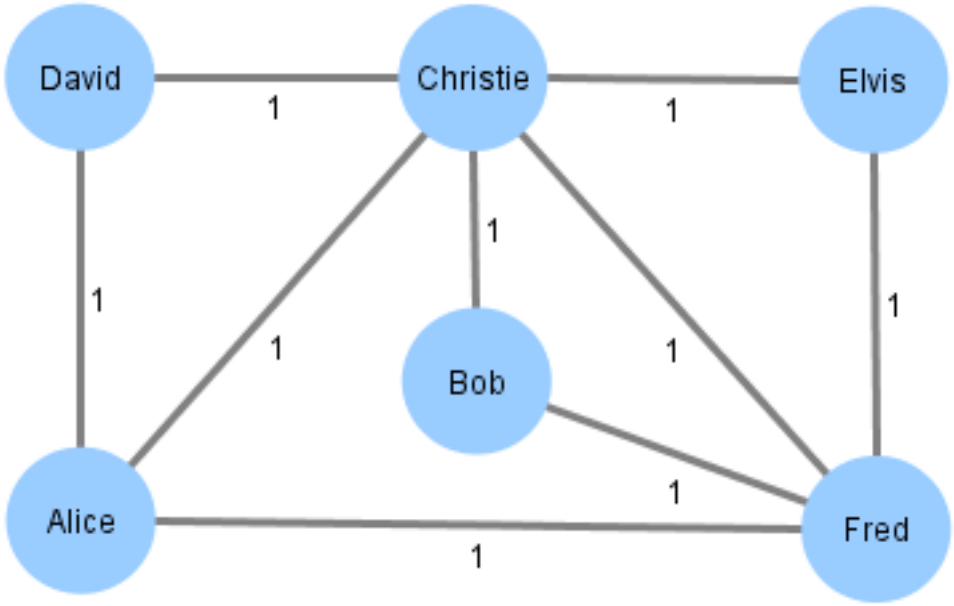
	RSTE	**SocialMF**	**SoRec**
Predicting $\mathcal{R}(u, i)$	$O(k \cdot \|T\|)$	$O(k)$	$O(k)$
Updating $U_{u,k}$	$O(k \cdot m \cdot \|T\|^2)$	$O(k \cdot m + \|T\|^2)$	$O(k \cdot (n + m))$
Updating $I_{i,k}$	$O(k \cdot n \cdot \|T\|)$	$O(k \cdot n)$	$O(k \cdot n)$
Updating $V_{v,k}$	-	-	$O(k \cdot n)$

For all of the social-aware models given here follows that complexity is linear in the dimension of latent features k when using appropriate caching strategies. Furthermore SocialMF and SoRec offer the advantage of directly modifying the user latent features towards socially connected users, so that the complexity of prediction does not deteriorate over PMF. Finally the complexity analysis also indicates that the incorporation of social information is handled most efficiently by SocialMF.

7 Tipping Weights of Trust

The proposed social-aware models base on the PMF framework and incorporate trust information by adjusting the active user's latent features (SoRec, SocialMF) or the predicted rating (RSTE) in the direction of socially connected users. In general, social networks do not allow the users to specify how much other users/friends are trusted. Also the relations are often undirected (e.g. Facebook, Google+, LinkedIn, Xing, etc.), so that trust matrices are typically binary and symmetric, as presented in table 2.3. Figure 7.1 illustrates the network of table 2.3. Since there is no specific trust information available to us, all edges have weight 1.

Figure 7.1: Network visualization of trust matrix given in table 2.3.



7.1 Equal Distribution

In the proposed models SocialMF and RSTE both normalize the influence of trusted users equally, such that all trusted users of the active user have similar impact on the resulting rating (RSTE) or the user's latent features (SocialMF). The trust weight $T_{u,v}$ between user u and user v therefore denotes as

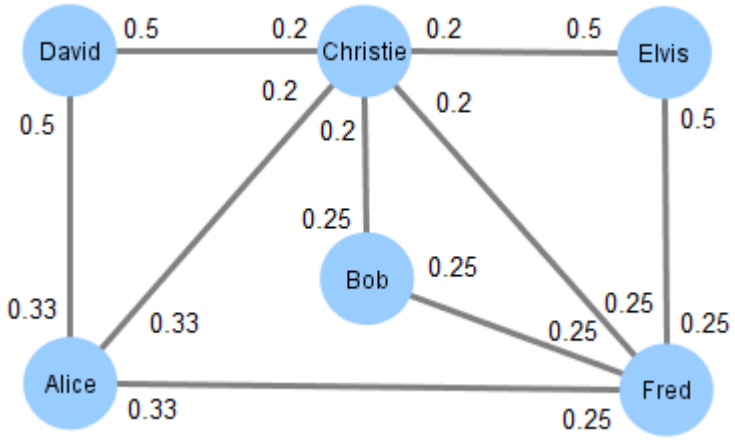
$$T_{u,v} = \frac{1}{|N(u)|} \tag{7.1}$$

which implies that

$$\forall u \in U : \sum_{v \in N(u)} T_{u,v} = 1 \tag{7.2}$$

The resulting edge weights for social connections of table 2.3 are given in figure 7.2.

Figure 7.2: Visualization of table 2.3 after application of equal trust weights.



We can rewrite the loss functions of SocialMF and RSTE to

$$\mathcal{L}(R,T,U,I)^{SocialMF} = \tag{7.3}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2$$

$$+ \frac{\lambda_T}{2}\sum_{u=1}^{n}(U_{u,\cdot} - \sum_{v \in N(u)} \mathbf{T_{u,v}} \cdot U_{v,\cdot})^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2$$

and

$$\mathcal{L}(R,T,U,I)^{RSTE} = \tag{7.4}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(\alpha \cdot U_{u,\cdot} \cdot I_{i,\cdot}$$

$$+ (1-\alpha)\sum_{v \in N(u)} \mathbf{T_{u,v}} \cdot U_{v,\cdot} \cdot I_{i,\cdot}))^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2$$

in order to receive frameworks for further experimentation with trust weights $T_{u,v}$. Additionally the SoRec loss function allows experimentation with trust weights, since here as well these can be adjusted before training the model:

$$\mathcal{L}(R,T,U,I,V) = \tag{7.5}$$

$$\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2$$

$$+ \frac{\lambda_T}{2}\sum_{u=1}^{n}\sum_{v=1}^{n} Y_{u,v}(\mathbf{T_{u,v}} - g(U_{u,\cdot} \cdot V_{v,\cdot}))^2$$

$$+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2 + \frac{\lambda_V}{2}||V||_F^2$$

7.2 Hubs and Authorities

The original SoRec model has already been proposed to make use of a specific trust weight, however, it lacks of an evaluation to what extent choosing an alternative trust weight does actually improve the accuracy of the prediction model.

The authors propose to decrease the trust value $T_{u,v}$ if u *trusts* a lot of users, and increase $T_{u,v}$ if v *is trusted by* a lot of users:

$$T_{u,v} = \sqrt{\frac{d^-(v)}{d^+(u) + d^-(v)}} \tag{7.6}$$

where $d^+(u)$ is the outdegree and $d^-(u)$ the indegree of node u. Since we do not consider directed graphs in our evaluation, we simplify this weight

to

$$T_{u,v} = \sqrt{\frac{|N(v)|}{|N(u)| + |N(v)|}} \tag{7.7}$$

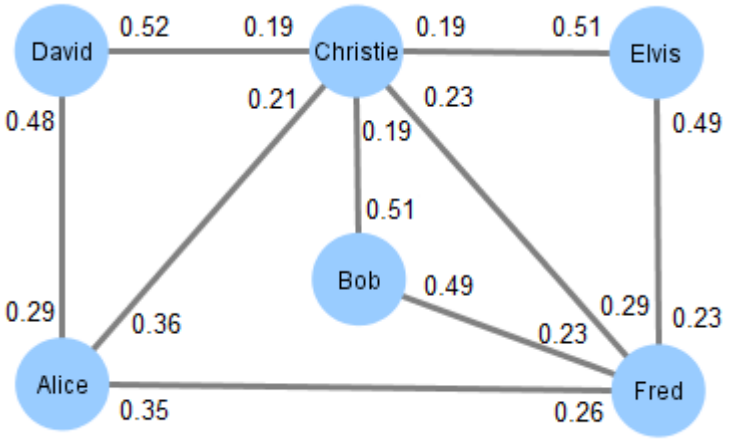
and refer to it as *HA**-weights, since the original directed version highlights **H**ubs and **A**uthorities of the network. Furthermore we propose *HA*-weights with additional normalization of sums of trust per user to 1 by again demanding

$$\forall u \in U : \sum_{v \in N(u)} T_{u,v} = 1 \tag{7.8}$$

in order to ensure model stability towards hyperparameters.

Figure 7.3 shows the sample trust matrix (table 2.3) after application of HU-weights.

Figure 7.3: Visualization of table 2.3 after application of HA-weights.



7.3 PageRank

The PageRank[7] algorithm is a well known method to aggregate weights from pure connections of the network. We suggest to make use of PageRank as a preprocessing step to identify opinion leaders in the network, and increase their weights compared to followers. There exists a vast number of other centralities in graph theory, however, we will pursuit PageRank here, as it is a good possibility to identify the sources of information. In comparison there is e.g. betweenness centrality[11] which measures how often a user is involved in shortest paths between other users. But betweenness

does more conflate information of different sources in the bridging nodes, and therefore these users might be less useful in terms of reliability. Contrairwise one can argue that users with higher betweenness are aware of several *opinions* and therefore their rating might be more funded, so that these users might also influence more than users with lower betweenness. In the domain of social networks [41] evaluate the predictive power of centralities and also combinations against each other. According to their results PageRank is performing well steadily also in combination with other centralities, so that we will focus on PageRank for our task, too.

Again we transfer PageRank to undirected graphs:

$$T_{u,v} = w(v) \tag{7.9}$$

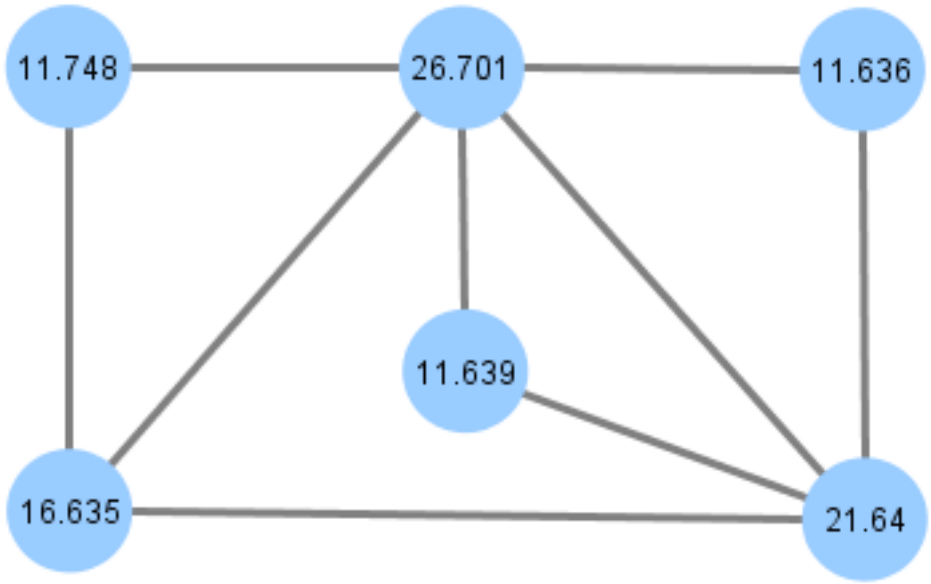
with

$$w(u) = \frac{1-d}{n} + d \sum_{v \in N(u)} \frac{w(v)}{|N(v)|} \tag{7.10}$$

where d is a damping factor, which we set to $d = 0.85$ since this is a typical choice for PageRank, also according to findings of [12].

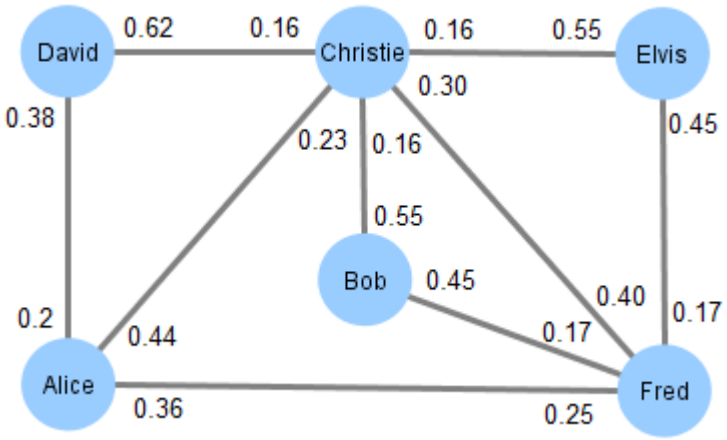
Figure 7.4 outlines the resulting PageRanks for the sample trust network of table 2.3.

Figure 7.4: PageRanks for users according to table 2.3 (note the network structure is equal to figure 7.2).



Again we demand trust weight sums per user to be equal 1, so that figure 7.5 shows the final weights for the sample network of table 2.3 after application of PageRank-weights.

Figure 7.5: Visualization of table 2.3 after application of PageRank-weights.



7.4 Social Regularization

In [25] the authors propose two further models which allow to weight trust by similarity of the active user and friends. First they introduce a so called *average-based regularization* which is, compared to PMF, an additional regularization term given as

$$\frac{\lambda_T}{2}\sum_{u=1}^{n}(U_{u,\cdot} - \frac{\sum_{v\in N(u)} sim(u,v)\cdot U_{v,\cdot}}{\sum_{v\in N(u)} sim(u,v)})^2 \tag{7.11}$$

where $sim(u,v)$ represents a similarity function between u and v. We can now rename $sim(u,v,)$ to the already familiar notation $T_{u,v}$ and, by again demanding the sum of trust weights per user to be equal zero, the situation

reduces to the weight-aware version of SocialMF as given in equation 7.3:

$$\begin{aligned}\mathcal{L}(R,T,U,I)^{SocialMF} = \\ &\frac{1}{2}\sum_{u=1}^{n}\sum_{i=1}^{m} X_{u,i}(R_{u,i} - g(U_{u,\cdot} \cdot I_{i,\cdot}))^2 \\ &+ \frac{\lambda_T}{2}\sum_{u=1}^{n}(U_{u,\cdot} - \sum_{v \in N(u)} \mathbf{T_{u,v}} \cdot U_{v,\cdot})^2 \\ &+ \frac{\lambda_U}{2}||U||_F^2 + \frac{\lambda_I}{2}||I||_F^2\end{aligned}$$

Second the authors present an *individual-based* regularization, where PMF is additionally regularized by

$$\frac{\lambda_T}{2}\sum_{u=1}^{n}\sum_{v \in N(u)} sim(u,v) \cdot (U_{u,\cdot} - U_{v,\cdot})^2 \tag{7.12}$$

Again we rename $sim(u,v)$ to $T_{u,v}$ which leads to

$$\frac{\lambda_T}{2}\sum_{u=1}^{n}\sum_{v \in N(u)} T_{u,v} \cdot (U_{u,\cdot} - U_{v,\cdot})^2 \tag{7.13}$$

Next we take $T_{u,v}$ into the brackets since $T_{u,v} = \sqrt{T_{u,v}}^2$

$$\frac{\lambda_T}{2}\sum_{u=1}^{n}\sum_{v \in N(u)} (\sqrt{T_{u,v}} \cdot (U_{u,\cdot} - U_{v,\cdot}))^2 \tag{7.14}$$

From the above term we derive a similarity matrix M^{IB} as

$$M^{IB} := (\sum_{v \in N(u)} \sqrt{T_{u,v}}) \cdot \mathbf{I} - \sqrt{T_{u,v}} \cdot \mathbf{I} \cdot A \tag{7.15}$$

so that we can express the individual-based model via the generic, social-aware matrix factorization model introduced in section 6.3 by

$$\mathcal{L}(R,T,U,I)^{IB} = \mathcal{L}(R,T,U,I)^{GEN} \tag{7.16}$$

with $M = M^{IB}$ in $\mathcal{L}(R,T,U,I)^{GEN}$ and $\mathcal{R}(u,i)$ the PMF prediction model.

In terms of their similarity function $sim(u, v)$ the authors propose Pearson product-moment correlation coefficient, which we can also apply to our $T_{u,v}$:

$$T_{u,v} = \frac{\sum_{i \in I^u \cap I^v} (R_{u,i} - \bar{R}_{u,\cdot}) \cdot (R_{v,i} - \bar{R}_{v,\cdot})}{\sqrt{\sum_{i \in I^u \cap I^v} (R_{u,i} - \bar{R}_{u,\cdot})^2} \sqrt{\sum_{i \in I^u \cap I^v} (R_{v,i} - \bar{R}_{v,\cdot})^2}} \tag{7.17}$$

Referring to section 3.1, where all efforts in collaborative filtering once started with PPMCC in 1994, we meet again in the above equation. Nevertheless the approach of weighting friends according to correlating rating patterns seems to be risky, since the intersection of rated items is already small among *all* users. Thus the intersection of a user's rated items with items rated by friends might rather vanish completely. Surprisingly the evaluation in [25] shows that using PPMCC in the average-based or individual-based model improves the RMSE of e.g. RSTE by up to 3.2%. In our evaluation, however, we find that most connections simply drop out since trust weights become zero when the intersection of rated items is empty, and thus accuracy worsens. For example, after application of PPMCC-weights to the Epinions dataset (cf. chapter 8) only $\approx 2\%$ of the connections are preserved.

Moreover the authors of [25] propose and evaluate weighting by cosine similarity, thus $T_{u,v}$ is computed as

$$T_{u,v} = \frac{\sum_{i \in I^U \cap I^V} R_{u,i} \cdot R_{v,i}}{\sqrt{\sum_{i \in I^U \cap I^V} R_{u,i}^2} \sqrt{\sum_{i \in I^U \cap I^V} R_{v,i}^2}} \tag{7.18}$$

In order to avoid complete vanishing of connections we further suggest to use Laplace smoothing. Also some of the extensions presented throughout section 3.1 (e.g. default voting, fixed profile means) could be applied to improve weighting trust based on the mere network structure and ratings.

7.5 Profile- and Item-Level Trust

Another two approaches for trust weights are given in [27]: *profile-level* and *item-level* trust. Both weights, however, do not postulate any information about the network, so that they only operate on the rating matrix itself.

First the authors define an indicator function $Correct(i, p, c)$ for item i, producer p and consumer c (where p and c are users of the recommender system) which states whether the absolute rating distance between p and c for i is smaller than a given ϵ:

$$Correct(i, p, c) \Leftrightarrow |p(i) - c(i)| < \epsilon \tag{7.19}$$

Next there is defined the $RecSet(p)$ of the producer p as the set of all tuples $(c, i) \in R$ for which p could be an indicator:

$$RecSet(p) = \{(c_1, i_1), ...(c_n, i_n)\} \tag{7.20}$$

Now for each observed rating $(c, i) \in R$ is added to another $CorrectSet(p)$ of user p if p actually *was* an indicator (i.e. $Correct(i, p, c)$ is *true*):

$$CorrectSet(p) = \{(c, i) \in RecSet(p) : Correct(i, p, c) \text{ is true.}\} \tag{7.21}$$

Profile-level trust of producer p is then defined as the percentage of correct indications of p among all tuples in the $RecSet(p)$ of p:

$$T^P(p) = \frac{|CorrectSet(p)|}{|RecSet(p)|} \tag{7.22}$$

The profile-level trust can be regarded as global metric per user (producer), which does not make a difference on the active user (consumer) or active item that the recommender is about to predict.

This drawback is tackled by *item-level* trust where the trust metric needs the active item as an argument, too:

$$T^I(p, i) = \frac{|\{(c, i_k) \in CorrectSet(p) : i_k = i\}|}{|\{(c, i_k) \in RecSet(p) : i_k = i\}|} \tag{7.23}$$

Unlike $T^P(p)$ which can be further handled in analogy to PageRanks and therefore be used in all proposed trust-aware models, the incorporation of $T^I(p, i)$ in social-aware matrix factorization can only be applied to the RSTE model, where i is a bound parameter at the time when accessing $T_{u,v}$. Remember RSTE incorporates trust in the prediction term, where a bound i is available, so that $T_{u,v}$ could be further parameterized by i. However, in SocialMF trust is processed in the regularization of U, thus $T_{u,v}$ can not be made dependent on i, since i would not be bound here. A similar situation holds for SoRec, where the trust matrix is factorized independently of an item i.

A last extension proposed in [27] is filtering trust by threshold, such that only trust weights $T^*_{u,v}$ greater than a given minimum Ω are considered:

$$T^*_{u,v} = (T_{u,v})^{Z_{u,v}} \tag{7.24}$$

where

$$Z_{u,v} = \begin{cases} 1 & \text{, if } T_{u,v} > \Omega \\ 0 & \text{, else.} \end{cases} \tag{7.25}$$

7.6 Summary

We demonstrated how the proposed social-aware matrix factorization models can be transformed in order to enable consideration of alternative trust metrics.

Furthermore we could show that the average-based regularization presented in [25] can be reduced to SocialMF, and also the individual-based regularization approach is expressible by our generic model derived in section 6.3. Thus both also provide room for experimentation with trust weights.

Next to already available equal weighting of trust among friends, we proposed PageRank to estimate weights based on the pure network structure. A similar approach is given in [24] where the authors reinforce hubs and authorities of the network. We also regarded weighting based on similar rating patterns, e.g. by correlation or vector similarity, as well as trust metrics which do not consider the network structure at all.

For parameterization of the weighting function with the active item we further reached limits of particular factorization models which do not incorporate trust within the prediction term.

8 Evaluation

In the following we focus on the evaluation of the proposed matrix factorization models on mainly 2 real life data sets. We compare accuracy in terms of RMSE and actual runtimes of the models to each other. Also we provide appropriate hyperparameters for the data sets, which have been determined by large-scaled cross-validations with numerous settings.

8.1 Synthesizing Data Sets

As a side note for testing correctness of model implementations, it can be helpful to synthesize data sets which perfectly fit a model's assumptions.

For example, the RSTE model would probably best factorize a rating matrix that is exactly the product of user and item latent features U and I, where for each of the n users u the following holds:

$$U_{u,\cdot} \sim \alpha U'_{u,\cdot} + (1-\alpha) \cdot T_{u,v} \sum_{v \in N(u)} U'_{v,\cdot} \tag{8.1}$$

Thus we would initialize U' with small random values, and iteratively adjust U as given above. After each iteration we set $U' \leftarrow U$, and perform until convergence. The dimension of latent features k within synthetization can be chosen equal to the evaluation for optimality. From the so yielded U together with e.g. Gaussian initialized I there can be computed a rating matrix for which when training the RSTE model with, the prediction error should go to zero when disabling regularization.

8.2 Real Life Data Sets

For the actual evaluation of the proposed models, however, we rely on real life data sets which were crawled from various social web sites.

Epinions[1] is a network where people can share reviews about various products, from cars over media to even office supply. For our evaluation we used the data provided by the Alchemy repository[2] which has also been referenced in [35], [34] and [16], however, there also exists another variant offered by Trustlet[3].

Furthermore we evaluate models on the Douban data set[4] which has been crawled by the authors of [25]. According to the Wikipedia article[5] Douban[6] *"is a Chinese social networking service website allowing registered users to record information and create content related to film, books, music, and recent events and activities in Chinese cities"*.

Lastly we are also aware of Flixster[7], a site where users can share movie reviews and provide ratings. The Flixster data set has been crawled and is provided[8] by the authors of SocialMF[16].

Table 8.1 shows statistics of the aforementioned data sets based on the rating matrices. All data sets provide ratings on a scale from 1 to 5. The number of rating users in Flixster is about three times the number of Epinions. However, there are only half the number of items in Flixster than in Epinions. With about 8 million ratings the density of the Flixster rating matrix is 0.1138%. In comparison the rating density of Epinions is only 0.0355%. In Epinions rating users averagely rated 12.2 items, while each item has been rated 5.5 times on average. Note that standard deviations from these averages are quite large, which suggests that Epinions is rather unbalanced in terms of numbers of ratings per user or item. However, this is not unusual for social networks, since in general there are few *power users* and many *occasional users*.

A similar pattern can be observed for Flixster as well, with a standard deviation of 225.8 from a ratings per user mean 55.5, and a standard deviation of 943.5 from a ratings per item mean 168.0. We assume that the number of ratings per item is also strongly connected with the popularity of an item.

With about 16.8 million ratings of ∼130k users for ∼60k items Douban is the most dense data set of all three. According to table 8.12 which compares runtimes of the models for a single training iteration with gradient descent

[1] `http://epinions.com`
[2] `http://alchemy.cs.washington.edu/data/epinions/`
[3] `http://www.trustlet.org/wiki/Epinions_dataset`
[4] `http://www.cse.cuhk.edu.hk/irwin.king/pub/data/douban`
[5] `http://en.wikipedia.org/wiki/Douban`
[6] `http://douban.com`
[7] `http://flixster.com`
[8] `http://www.cs.sfu.ca/~sja25/personal/datasets/`

it is rather unfeasible to find optimal hyperparameters already for Flixster, as the following estimation will show:

Assume we would like to check performance of a model with 3 hyperparameters $\lambda_A, \lambda_B, \lambda_C$, and therefore choose 5 alternative values per hyperparameter, e.g. $\lambda_A, \lambda_B, \lambda_C \in \{100, 50, 10, 1, 0.1\}$. Let the maximum number of iterations t be 1000. Under a 5-fold cross validation protocol we would have to compute 3 hyperparameters $\times$ 5 values $\times$ 5 folds $\times$ 1000 iterations, which results in a total of 75,000 iterations. A single iteration for 80% of the ratings in Flixster takes e.g. 50 seconds, which is already optimistic compared to findings in table 8.12. Followingly the given evaluation scenario with 75,000 iterations would take 43 days.

For optimal model performance, which is essential for a well-founded evaluation, there are needed several more attempts with alternative hyperparameter values. Note that to the above runtime estimation there add up about 3 days when only increasing the number of values for a single hyperparameter by 1.

Table 8.1: Rating profiles of the data sets.

	douban	**douban***	**epinions**	**flixster**
users[1]	129,490	129,457	46,934	147,612
items	58,541	23,090	104,357	48,794
ratings	16,830,839	*1,000,000*	574,664	8,196,077
rating scale	1 to 5	1 to 5	1 to 5	1 to 5
ratings per user[1]				
min	1	1	1	1
max	6,328	25	847	30,977
mean	130.9	7.7	12.2	55.5
st.dev.	190.2	3.1	27.5	225.8
ratings per item				
min	1	1	1	1
max	49,504	5,680	1,992	34,791
mean	287.5	43.3	5.5	168.0
st.dev.	1541.2	189.5	22.4	943.5
density				
$r/(n \times m)$	0.2203 %	0.0335 %	0.0117 %	0.1138 %

Evaluations of different folds and settings can be scheduled in parallel, so that the absolute execution time for such an evaluation scenario can be decreased. However, since for Flixster there were not yet found reliable

[1] users with at least one rating

hyperparameters for *all* proposed models, we will not discuss or report qualitative results on this data set in favor of fairness.

Since the number of ratings in Douban is even twice the number of Flixster, we decided to derive a subsampled Douban* set with $1,000,000$ ratings, where we (1) randomly picked a user and (2) selected one of this user's ratings, such that the number of rating users is preserved while leveling off the unsteady rating frequencies among users. The resulting rating matrix is three times denser than Epinions. Another difference of Douban* and Epinions is the inverted shape of user- and item dimension: in Douban* there are more users than items, while for Epinions the number of users is smaller than the number of items.

Thus we are equipped with two manageable data sets of different characteristics on which we can perform the evaluation on.

Table 8.2 furthermore outlines statistics of the trust matrix for the data sets. Since Douban and Flixster sets only contain undirected information about relations, we decided to take Epinions as an undirected network for better comparison, too. The standard deviations for social connections per user also suggests the co-existence of power users and occasional users throughout the networks.

Table 8.2: Network profiles of the data sets.

	douban	**douban***	**epinions**	**flixster**
users[2]	111,210	111,210	75,880	786,936
connections	855,890	855,890	405,836	5,897,324
connections per user[2]				
min	1	1	1	1
max	986	986	3,044	1,045
mean	15.4	15.4	10.7	15.0
st.dev.	30.4	30.4	43.0	42.3

Among all three data sets the commitment of users to social activity also seems to be rather equivalent, since means and standard deviations are quite similar. Thus we suspect that users tend to create their social environment by recurring patterns independent of the domain of items, e.g. with the objective to be in the right loop when it comes to interesting updates. Followingly, as a next step towards exploitation of social information those patterns should be worked out in order to model them, e.g. within the matrix factorization framework. The proposed trust weighting by PageRank (cf. section 7.3) can be seen as a first step into this direction, since it tries

[2]users with at least one connection in the social network

to identify opinion leaders of the network - and *adding opinion leaders to social environment* seems to be an obvious behavioral pattern of users.

8.3 Evaluation Protocol

For the evaluation we split each data set into a *test set* of 50,000 randomly picked ratings and a *validation set* containing the rest of the ratings. On the validation set we perform 5-fold cross-validation to find optimal hyperparamaters, while the test set is held back to evaluate the so determined hyperparameters on.

For cross-validation the validation set is split into 5 folds, such that the number of ratings is equal among all folds. Now for each fold the remaining 4 folds are condensed to a *training set* which serves as input data for gradient descent. After each iteration of the learning algorithm both the root mean square error (RMSE) on the traning set as well as the RMSE on the validation fold are determined, where the RMSE on any subset R' of the observed ratings R is computed as

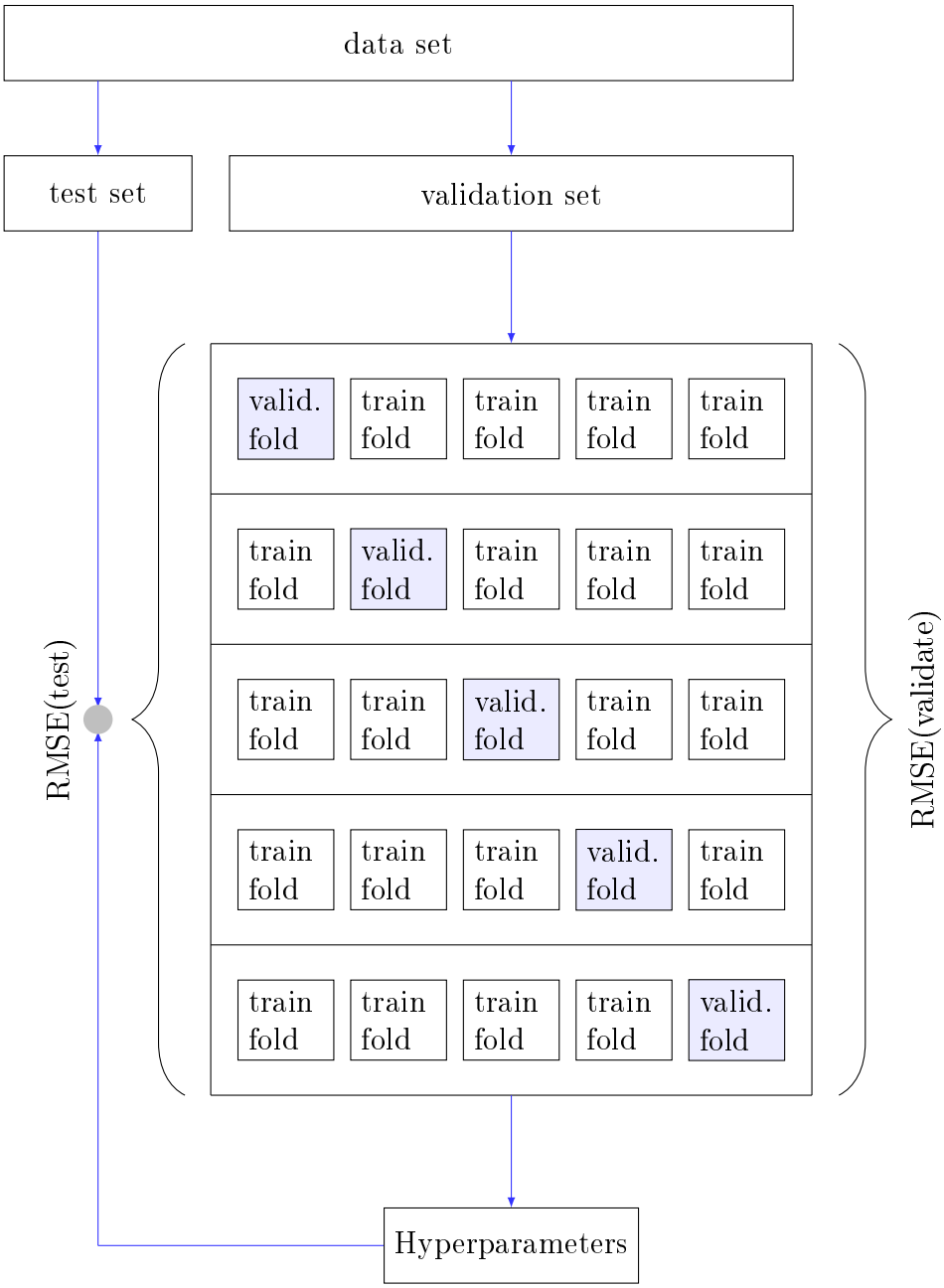
$$RMSE := \sqrt{\frac{1}{|(u,i) \in R'|} \sum_{(u,i) \in R'} \mathcal{P}(u,i)^2} \tag{8.2}$$

We define optimal hyperparameterization, learn rate and number of iterations for when the averaged RMSE among all 5 validation folds is minimal. We refer to this averaged RMSE as *RMSE(validate)* or *validation error*.

Now for the optimal parameterization obtained from cross-validation we would like to check a model's stability to actually unseen ratings. Therefore we set up 5 models with these optimal parameters and train them each with one of the 5 different training sets already used in cross-validation. We stop training after optimal number of iterations and determine the RMSE for the models on the test. The average RMSE of these 5 models on the test set is referred to as *RMSE(test)* or *test error*. Figure 8.1 further outlines the described evaluation process.

Since hyperparameters are optimized towards the validation folds during cross-fold validation, a differently biased test set may degenerate a models performance. Thus the test error heavily depends on the selection of the test set. In order to not accidentally favor one model over the other here, we decided to not fix a specific test set over all models, but always resample the 50k test ratings before sweeping through an evaluation process per model.

Figure 8.1: Schematic representation of the evaluation protocol.



8.4 Epinions

Table 8.3 provides an overview about the hyperparameter settings which were found to achieve best results during cross-validation on the validation set according to the aforementioned evaluation protocol. For hyperparameter search the models were set up with latent feature dimension $k = 5$.

Table 8.3: Applied hyperparameters, determined on Epinions for $k = 5$.

	λ_U	λ_I	λ_J	λ_T	λ_V	α
PMF	7	7	-	-	-	-
SVD++	7	7	15	-	-	-
SoRec	3	3	-	10	10	-
RSTE	3	3	-	-	-	0.4
SocialMF	3	3	-	7	-	-

Note that the optimal α for RSTE is 0.4 on Epinions, which suggests that friends represent 60% of a users rating here.

Table 8.4 summarizes the results of the factorization models. With an RSME of 1.096 SVD++ achieves best results on the validation set, however, the situation changes when predicting the test set. From the test error we can draw conclusions about the robustness of models, which is also a desirable property in real life recommender systems. On the smaller Epinions data set SVD++ seems to lack here, since it shows the maximum deterioration of 0.041 among all models from the validation error. The more robust RSTE model stands its ground and achieves the best test error with an RMSE of 1.126. Since SVD++ is known as state-of-the-art extension to matrix factorization doubts about this finding are welcome. We might argue that the test set for the evaluation of SVD++ here has been chosen unluckily. However, when regarding the results for $k = 10$ and $k = 20$, where for each row in the table a freshly sampled test set has been used according to the evaluation protocol, SVD++ again shows very high deterioration from the validation error, while RSTE could always take the lead on the test error.

For $k = 10$ and $k = 20$, however, we reused hyperparameters determined for $k = 5$ as given in table 8.3. In general, the errors did rather worsen here, which suggest that increasing the number of latent features leads to overfitting, so that hyperparameters have to be adjusted. Since time and computational resources were restricted we did not further investigate if an increased number of latent features with optimal hyperparameters would lead to better results, however, [16] and [23] present contradictory findings

Table 8.4: Model performances on Epinions for $k \in \{5, 10, 20\}$.

		valid. error	test error	learn rate	iterations
k=5					
	PMF	1.138	1.176	0.005	131
	SVD++	1.096	1.137	0.0005	290
	SoRec	1.112	1.138	0.0005	603
	RSTE	1.104	1.126	0.0005	883
	SocialMF	1.125	1.156	0.0005	1569
k=10					
	PMF	1.142	1.180	0.005	120
	SVD++	1.096	1.142	0.0005	270
	SoRec	1.119	1.142	0.0005	552
	RSTE	1.106	1.130	0.0005	811
	SocialMF	1.130	1.151	0.0005	1338
k=20					
	PMF	1.140	1.185	0.005	147
	SVD++	1.098	1.143	0.0005	255
	SoRec	1.122	1.148	0.0005	538
	RSTE	1.108	1.129	0.0005	782
	SocialMF	1.128	1.151	0.0005	2000

about the effects of increasing the number of latent features on the Epinions data set. In this regard, tables 8.5 and 8.6 summarize their RMSE errors, which were both collected via 80%/20% cross-validation and can thus be well compared to our validation error.

Table 8.5: Model performances from [23] for $k = 5$, $\lambda_U = \lambda_I = 0.001$.

	valid. error	test error
PMF	1.1826	-
SoRec	1.1530	-
RSTE	1.1346	-

We conclude that hyperparameter search is a critical task for successful matrix factorization models, since both [23] and [16] report worse RMSEs for PMF, SoRec and RSTE — probably not only due to the fact that they applied *stochastic* gradient descent. Only for SocialMF its authors [16] report an RMSE which even improves over our results for SVD++.

Table 8.6: Model performances from [16] for $k = 5$, $\lambda_U = \lambda_I = 0.1$.

	valid. error	**test error**
PMF	1.175	-
RSTE	1.145	-
SocialMF	1.075	-

However, our evaluation does not confirm such a performance of SocialMF as outlined in table 8.4, eventually due to the fact that we interpret trust connections as undirected on this data set, too.

We also implemented the memory-based PPMCC[33] approach presented in section 3.1, as well as the fix-means-to-mean-rating extension[39] described at the end of section 3.1, which we refer to as PPMCC++. Since for these there are no hyperparameters necessary, we skipped cross-validation and report results of repeated evaluations on the test set. Table 8.7 presents the results for PPMCC and PPMCC++ on Epinions. Though both lose to matrix factorization models in general, PPMCC++ slightly improves over PPMCC in terms of the RMSE here.

Table 8.7: Memory-based performances on Epinions.

	valid. error	**test error**
PPMCC	-	1.403
PPMCC++	-	1.399

Furthermore we provide runtimes of a single iteration over the training set within cross-validation, as well as the computation of the RMSE on the validation fold, which highly depends on a model's prediction speed. The runtimes were gathered on an Intel Core i7-2600 CPU clocked at 3.4 GHz. We implemented in Java and performed the evaluation on a Windows 7 64-bit Java Virtual Machine packed with 8 GB memory. Table 8.8 summarizes the measured runtimes for all models also with different dimensions of latent features k. Note that we made use of caching according to section 6.4 where applicable, thus we speeded up SocialMF by 60% and RSTE by even 93% on Epinions compared to findings in [16].

For PMF, SoRec and SocialMF prediction runtimes are equal, which was expected and confirms our complexity analysis (cf. table 6.2). For RSTE the prediction complexity also depends on trusts, so that it is typically slower than other models. Only SVD++ is slower here, since on the one

Table 8.8: Model runtimes on Epinions for $k \in \{5, 10, 20\}$.

		iteration over train set (ms)	**evaluation of valid. fold (ms)**
k=5			
	PMF	397	37
	SVD++	3,760	194
	SoRec	1,302	34
	RSTE	2,427	137
	SocialMF	947	36
k=10			
	PMF	592	42
	SVD++	7,408	317
	SoRec	1,906	40
	RSTE	4,357	195
	SocialMF	1,499	39
k=20			
	PMF	878	50
	SVD++	14,319	584
	SoRec	3,157	50
	RSTE	8,305	266
	SocialMF	2,663	48

hand there cannot be shared knowledge via caching between different users. On the other hand a user has averagely more rated items than connected users, which further charges SVD++ over social-aware models. For model training this drawback affects even more, so that again SVD++ appears slowest. Among social-aware models SocialMF takes the lead with ∼2.6 seconds per iteration over the training set.

However, since the optimal number of iterations for SocialMF is quantified by 1569 in our evaluation, the total training phase takes about 70 minutes for this model. In comparison RSTE only takes about 36 minutes, since there are only 883 iterations necessary. For SoRec there are needed 13 minutes for its 603 iterations, while SVD++ bills ∼18 minutes for 290 iterations.

Figures 8.2 to 8.10 present details of cross-validating the proposed models with $k = 5$. For all plots we already averaged the results among the 5 validation folds. There are presented courses of validation errors during the training phase for selected settings. For PMF we also check whether statements about hyperparameters can be made based on the traning error.

Figure 8.2: Validation error of PMF on Epinions. The best RMSE has been found for $\lambda_U = \lambda_I = 7$ after 131 iterations.

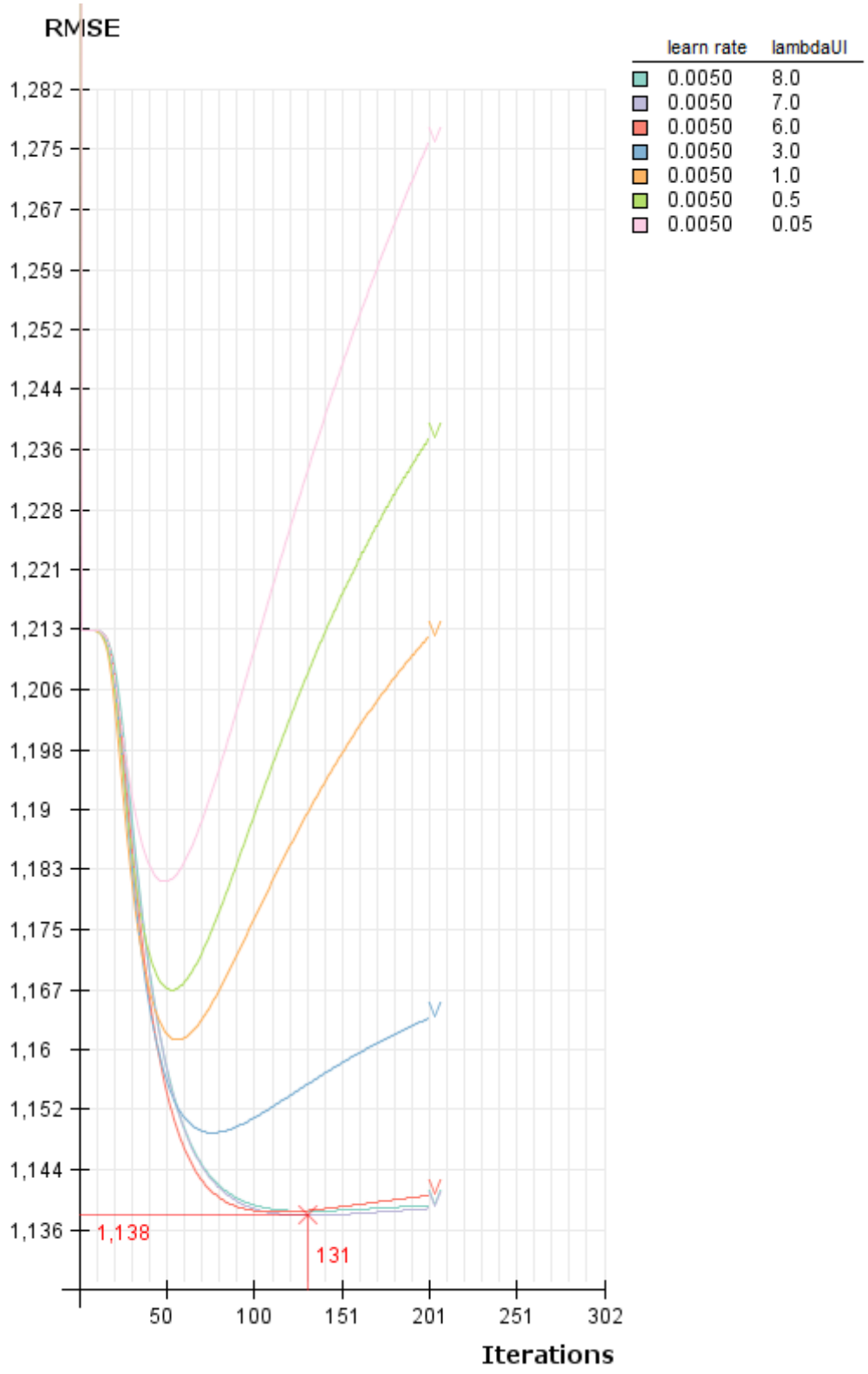


Figure 8.3: Training error of PMF on Epinions. At the time of best validation error the traning error is at about 0.98 for $\lambda_U = \lambda_I = 7$.

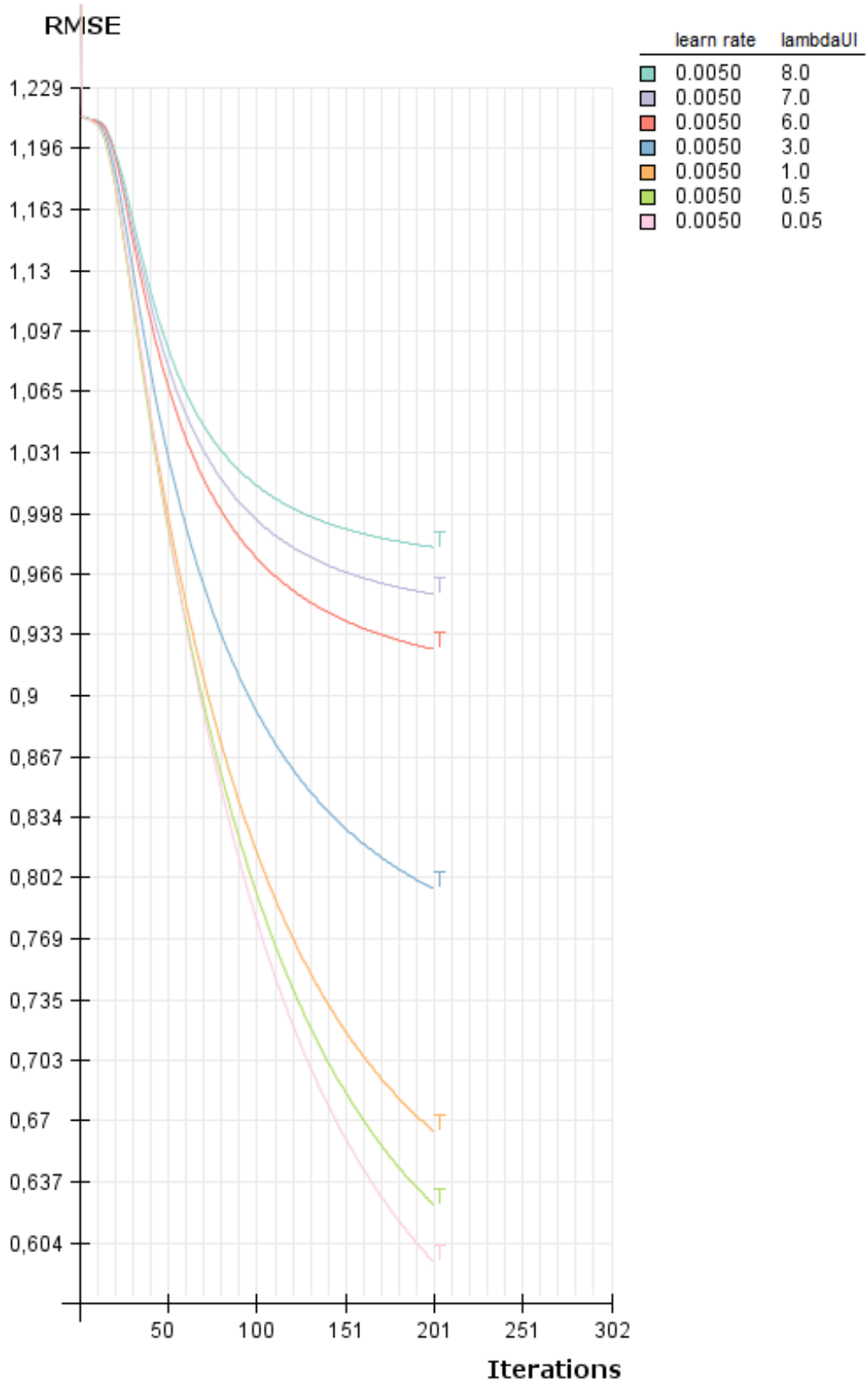


Figure 8.4: Difference between validation and training error of PMF on Epinions. Unfortunately we can not make a statement about optimality of hyperparameters based on the difference of validation and training error.

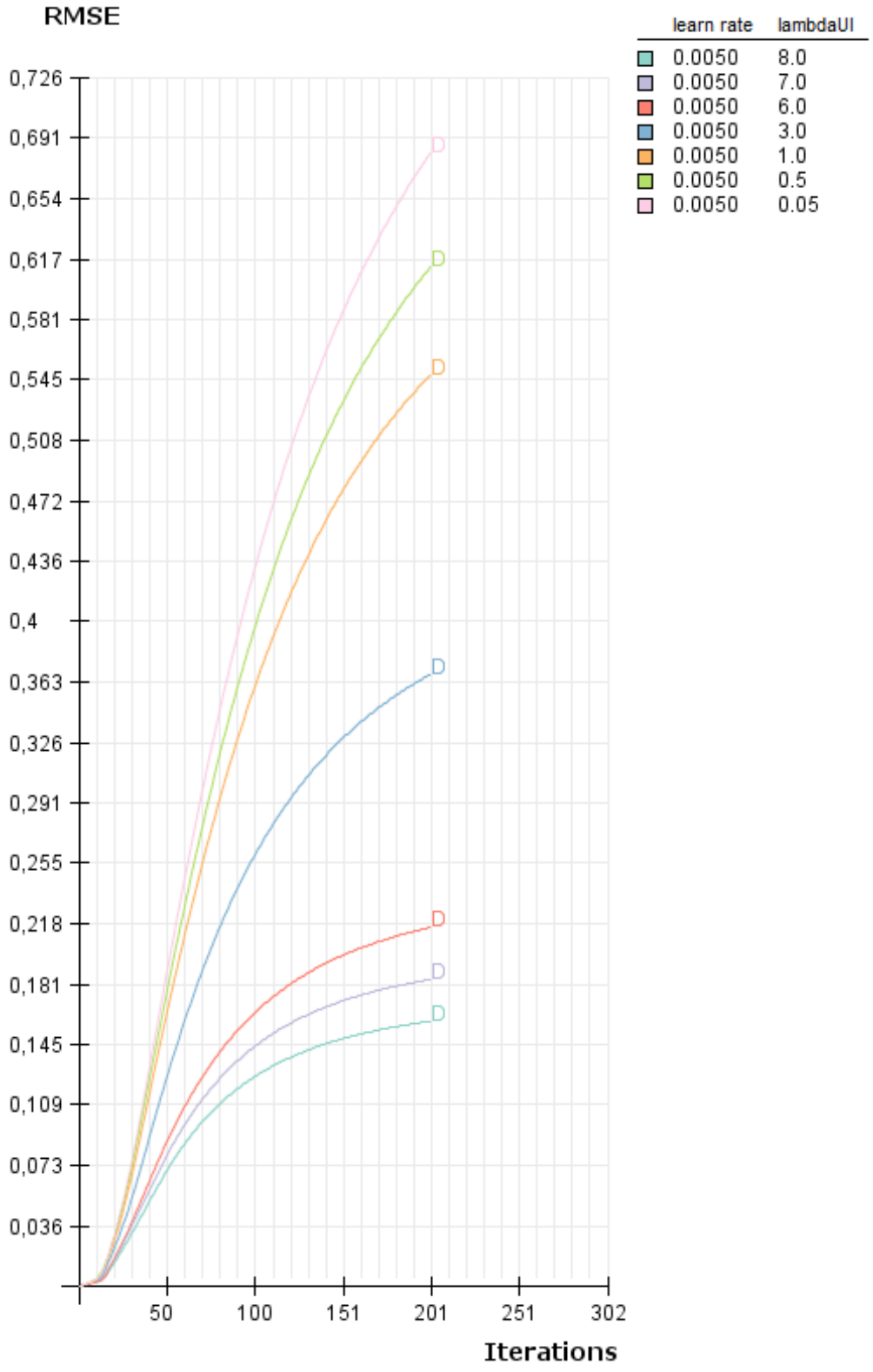


Figure 8.5: Validation error of SVD++ on Epinions for fixed $\lambda_U = \lambda_I = 7$. The model performs best for $\lambda_J = 15$ after 290 iterations.

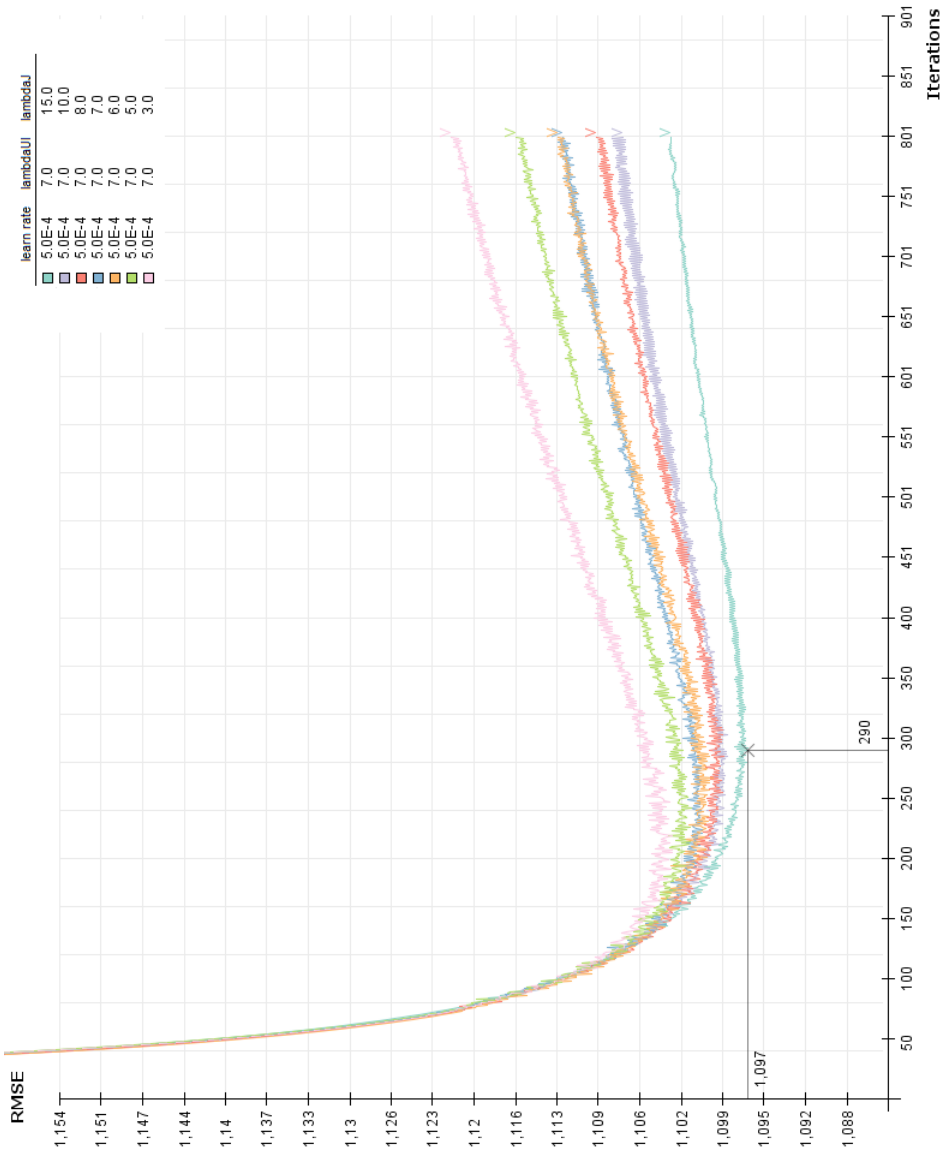


Figure 8.6: Validation error of SVD++ on Epinions for fixed $\lambda_J = 15$. The model performs best for $\lambda_U = \lambda_I = 7$.

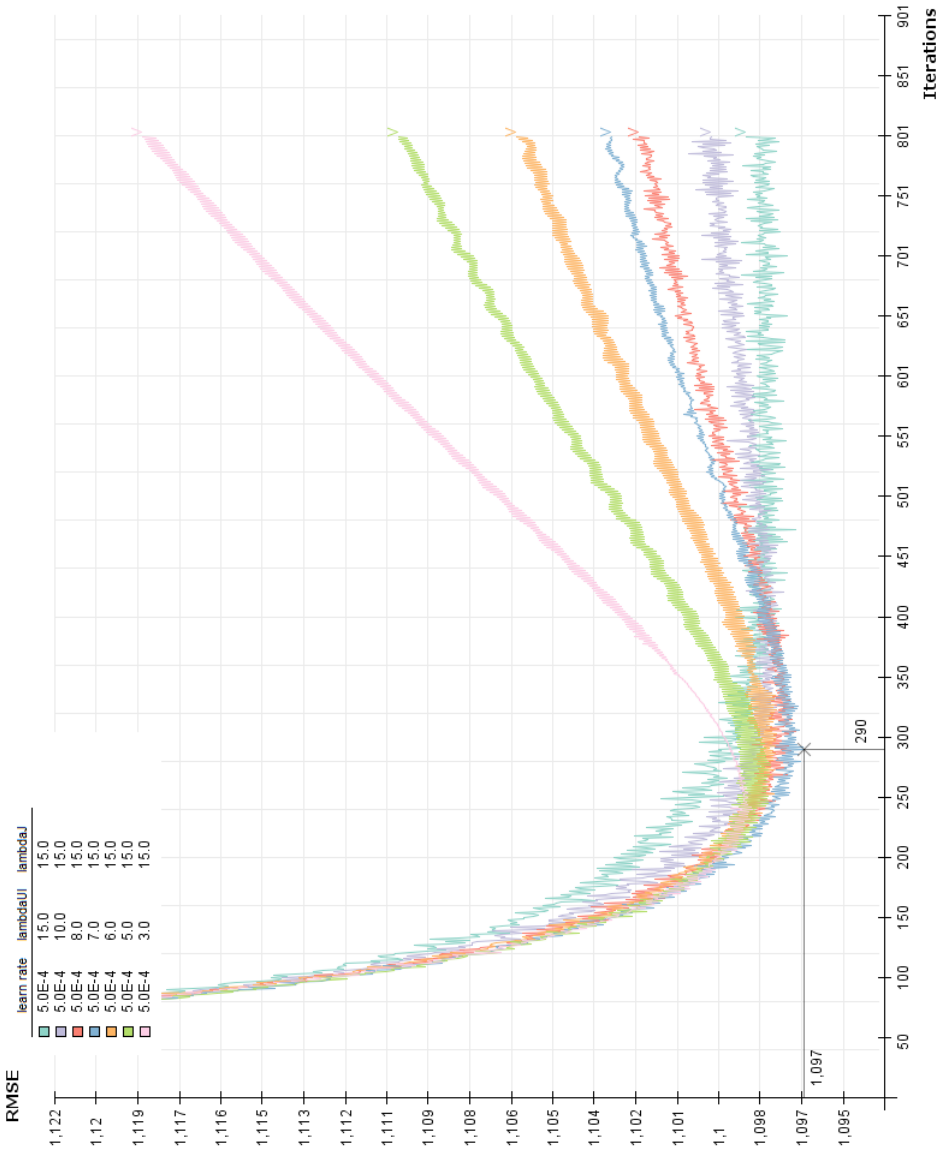


Figure 8.7: Validation error of SoRec on Epinions. The model performs best for $\lambda_T = \lambda_V = 10$ after 603 iterations.

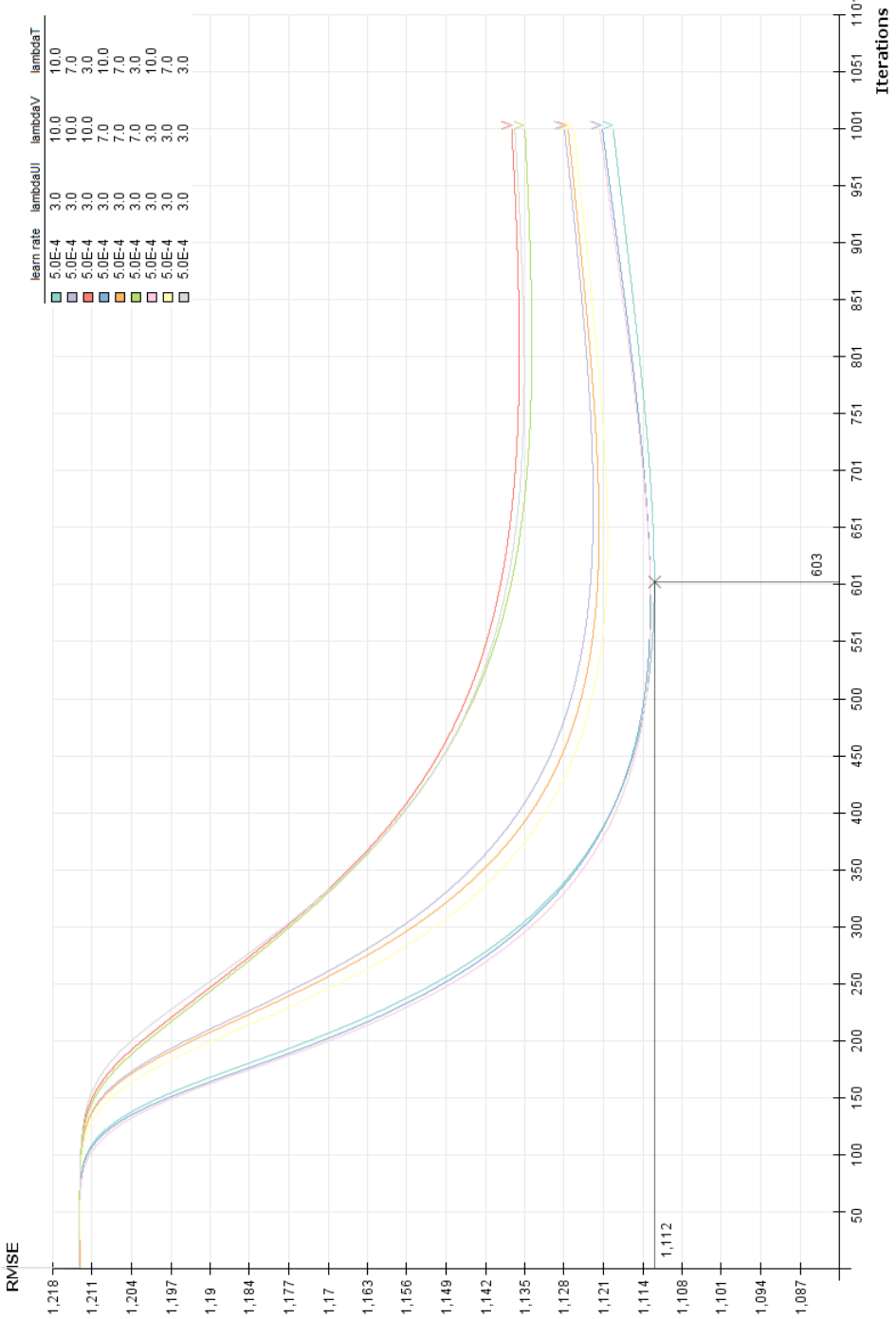


Figure 8.8: Validation error of RSTE on Epinions with fixed $\lambda_U = \lambda_I = 3$. The model performs best for $\alpha = 0.4$ after 883 iterations.

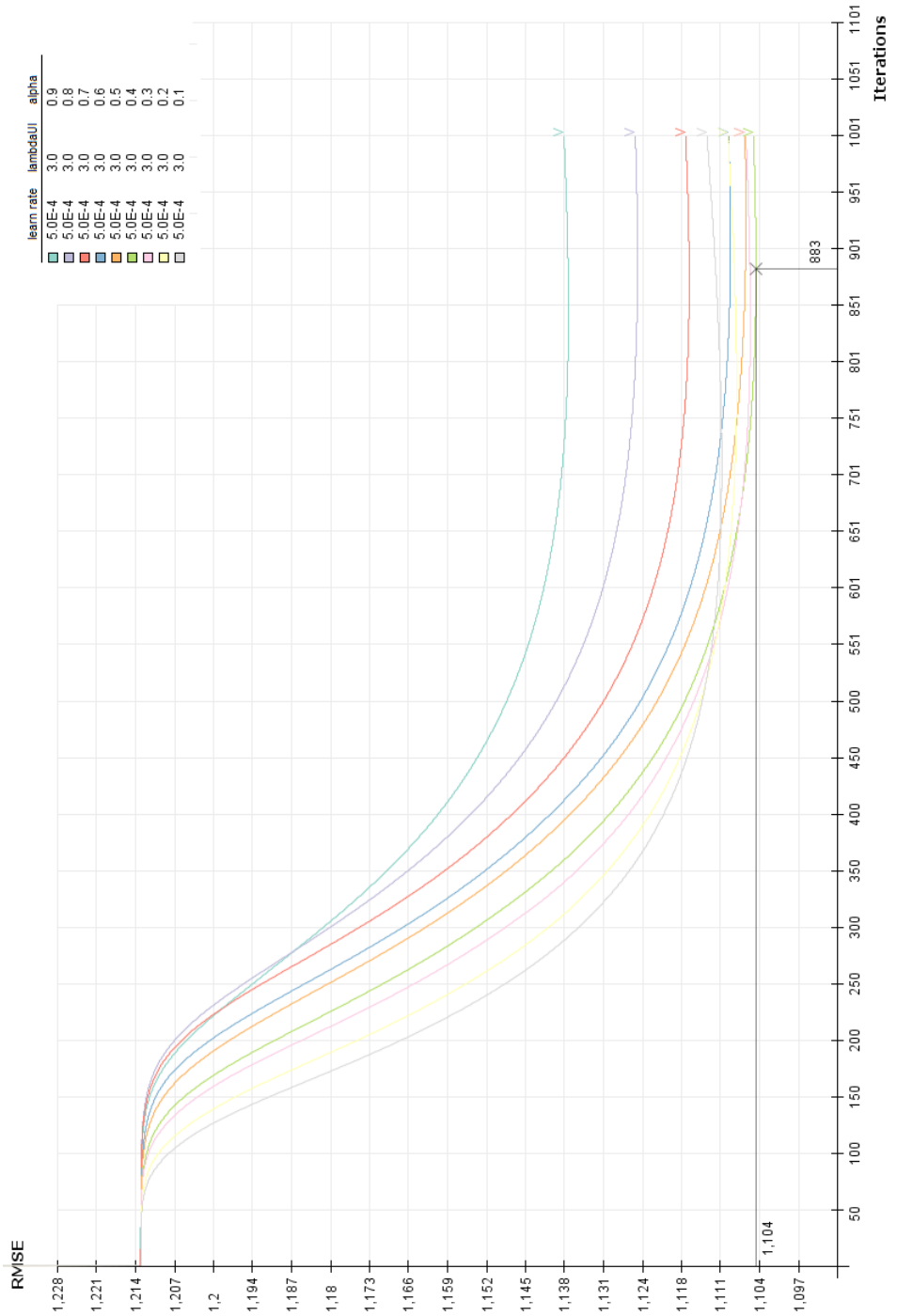


Figure 8.9: Validation error of RSTE on Epinions with fixed $\alpha = 0.4$. The model performs best for $\lambda_U = \lambda_I = 3$.

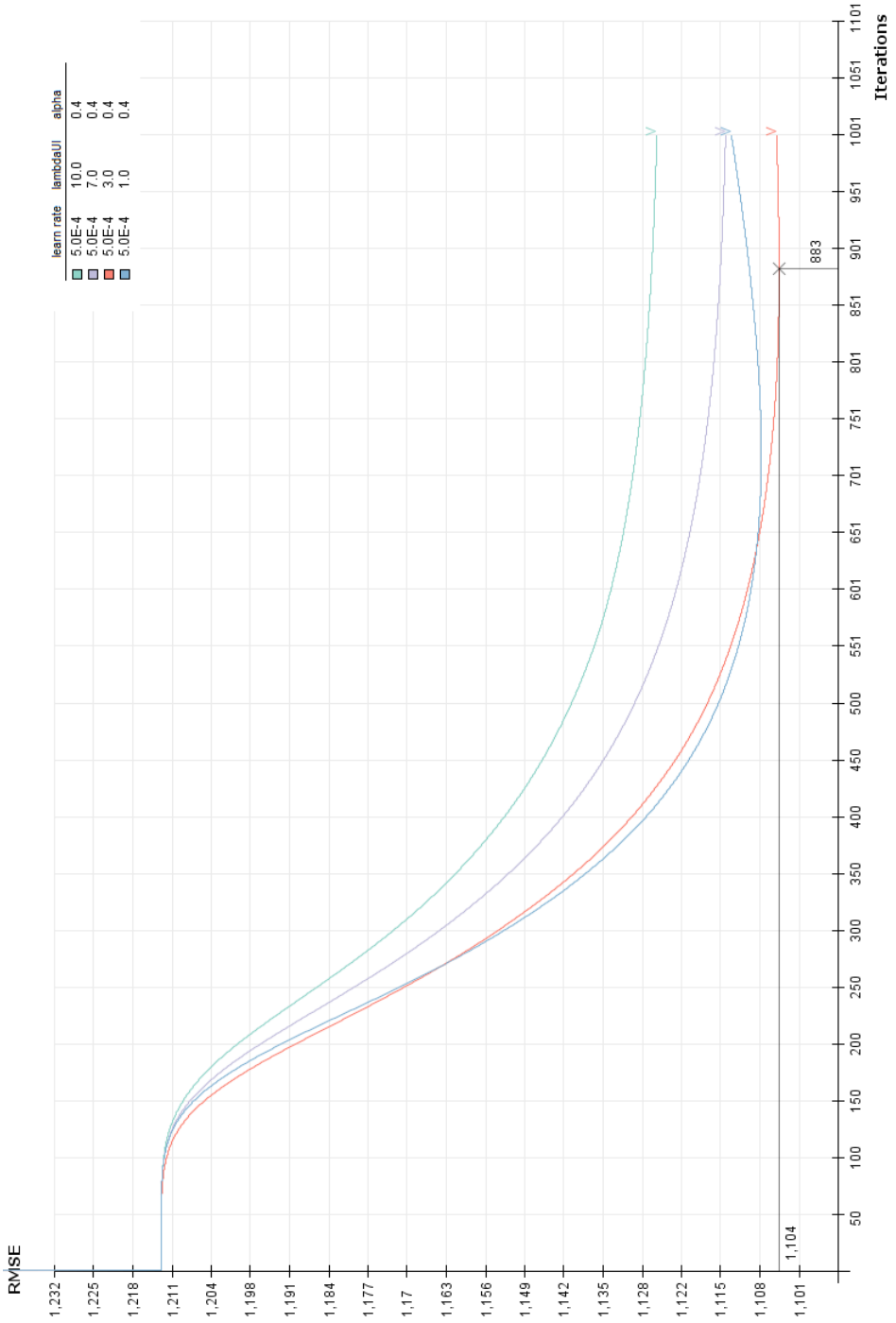
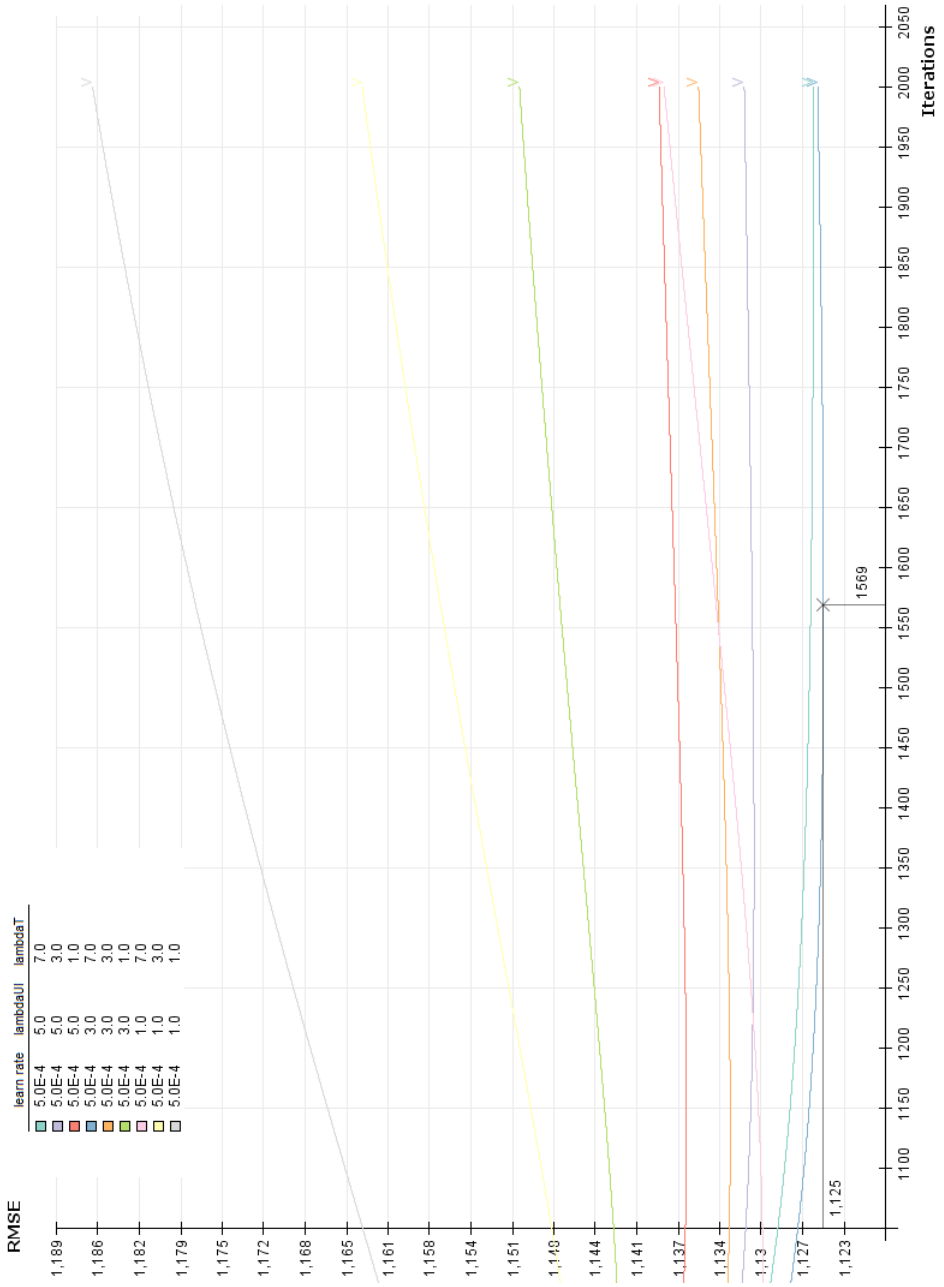


Figure 8.10: Validation error of SocialMF on Epinions. The model performs best for $\lambda_U = \lambda_I = 3$ and $\lambda_T = 7$ after 1569 iterations.



8.5 Douban

We further evaluated the proposed models on the subsampled Douban* data set. Table 8.9 summarizes validation and test errors for dimensions of latent features $k = 5$ and $k = 10$.

Table 8.9: Model performances on Douban* for $k = \{5, 10\}$.

		valid. error	**test error**	**learn rate**	**iterations**
k=5					
	PMF	0.853	0.856	0.005	812
	SVD++	0.809	0.814	0.0005	773
	SoRec	0.856	0.857	0.0005	565
	RSTE	0.827	0.832	0.0005	862
	SocialMF	0.828	0.828	0.0005	3000
k=10					
	PMF	0.853	0.860	0.005	683
	SVD++	0.812	0.821	0.0005	282
	SoRec	0.860	0.859	0.0005	572
	RSTE	0.828	0.832	0.0005	826
	SocialMF	0.824	0.823	0.0005	3000

On Douban* SVD++ is more robust than on the smaller Epinions set, so that it shows best RMSEs on the validation and test set among all models. SoRec does not improve over PMF, while SocialMF achieves better results than RSTE in terms of RMSE. However, the runtime of SocialMF degenerates, since after 3000 iterations we did not reach convergence yet. Increasing the learn rate is also a problem because the model does not converge in later iterations - an adaptive learn rate might improve the situation here. For our evaluation we decided to keep the learn rate of 0.0005 and stop after 3000 iterations.

Again large-scaled hyperparameter searches have been performed per model for $k = 5$ on Douban*. Table 8.10 presents the results, which were also reused to compute errors for $k = 10$ as presented in table 8.9. As for Epinions, increasing the number of latent features of models in Douban* leads to overfitting, so that convergence is reached after fewer iterations, but results rather deteriorate.

Table 8.10: Applied hyperparameters, determined on Douban* for $k = 5$.

	λ_U	λ_I	λ_J	λ_T	λ_V	α
PMF	7	7	-	-	-	-
SVD++	10	10	20	-	-	-
SoRec	3	3	-	10	1	-
RSTE	1	1	-	-	-	0.2
SocialMF	3	3	-	20	-	-

The optimal α for RSTE is even 0.2 on RSTE, so that friends represent 80% of a users rating here.

Table 8.11 further provides an overview about averaged performances of memory-based methods PPMCC and PPMCC++ on test sets. While both are again outperformed by model-based approaches, on Douban* the fix-means extension can not improve over PPMCC.

Table 8.11: Memory-based performances on Douban*.

	valid. error	**test error**
PPMCC	-	0.952
PPMCC++	-	0.953

Moreover, in figures 8.11 to 8.19 we present details about the validation error of models during cross-validation on Douban*.

Figure 8.11: Validation error of PMF on Douban*. The model performs best for $\lambda_U = \lambda_I = 7$ after 812 iterations.

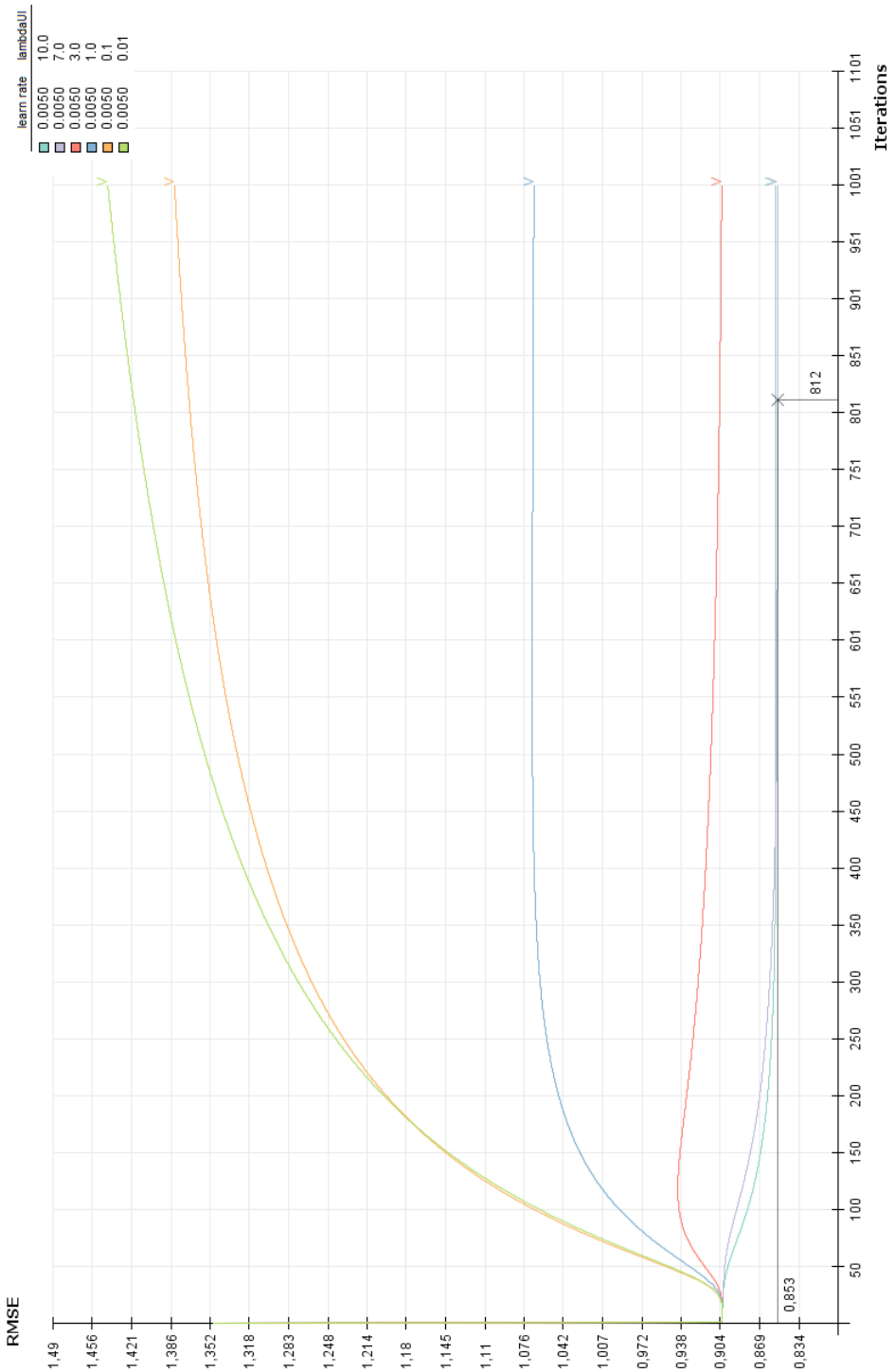


Figure 8.12: Training error of PMF on Douban*.

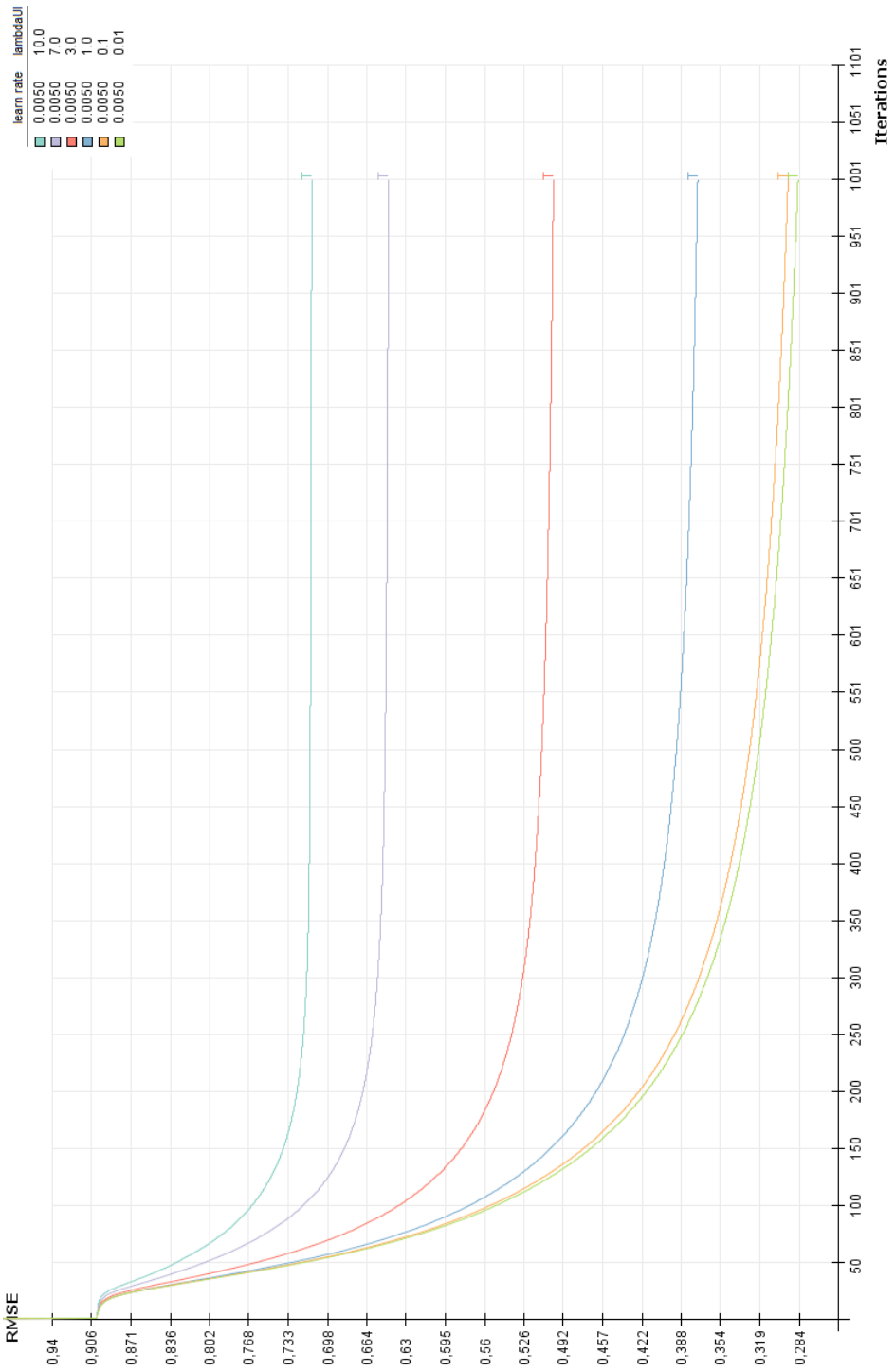


Figure 8.13: Difference between validation and training error of PMF on Douban*. Again we can not clearly say that the difference of errors bears obvious information about optimality of hyperparameters, however, it might be interesting that both Douban* and Epinions hyperparameter settings were optimal for PMF when the difference of errors converged at about 0.2.

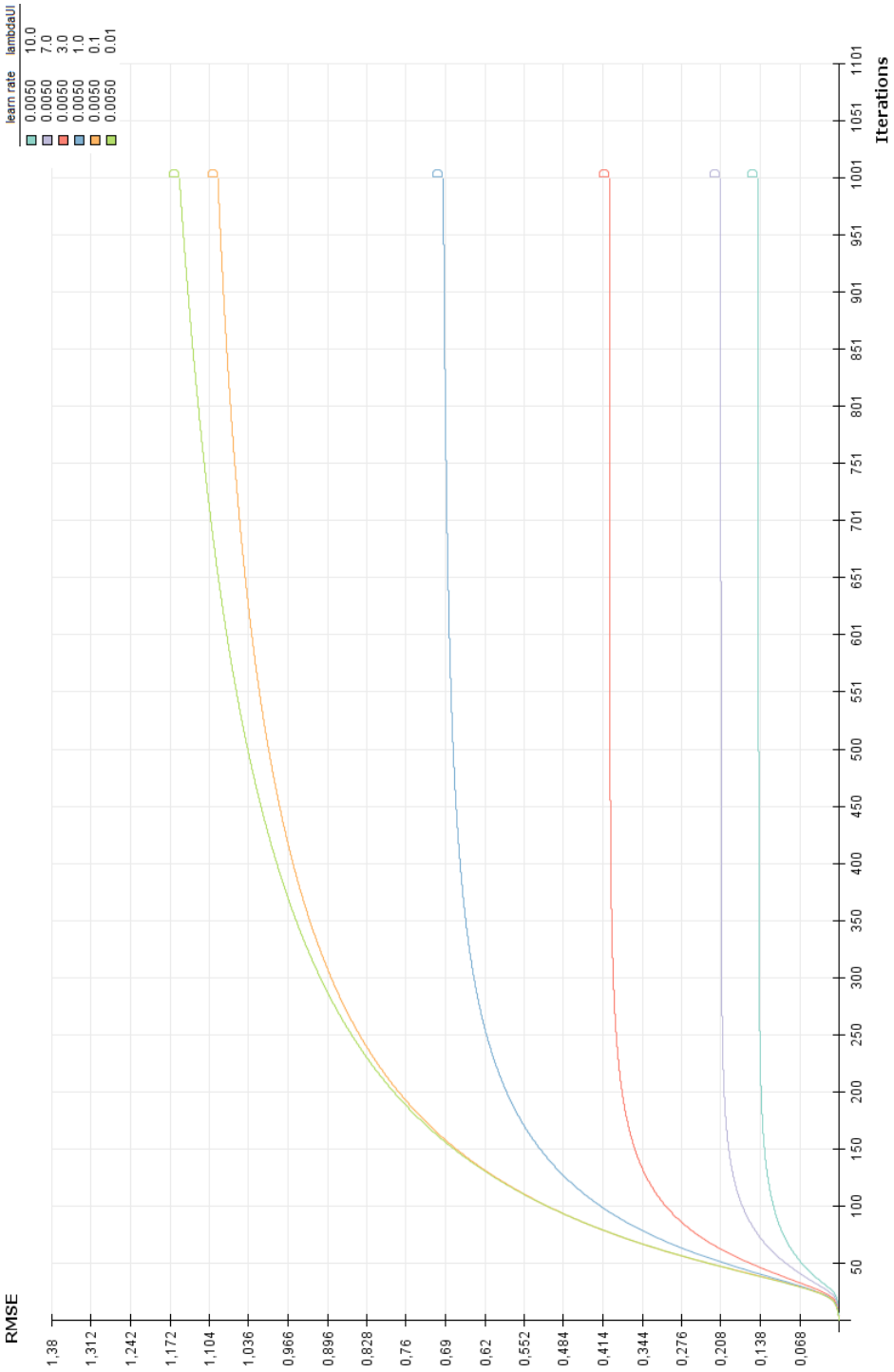


Figure 8.14: Validation error of SVD++ on Douban* with fixed $\lambda_J = 20$. The optimal setting has been found for $\lambda_U = \lambda_I = 10$ after 773 iterations. Note that also for SVD++ a non-adaptive learn rate is already critical, since already after 100 iterations the error starts to flicker around the optimum.

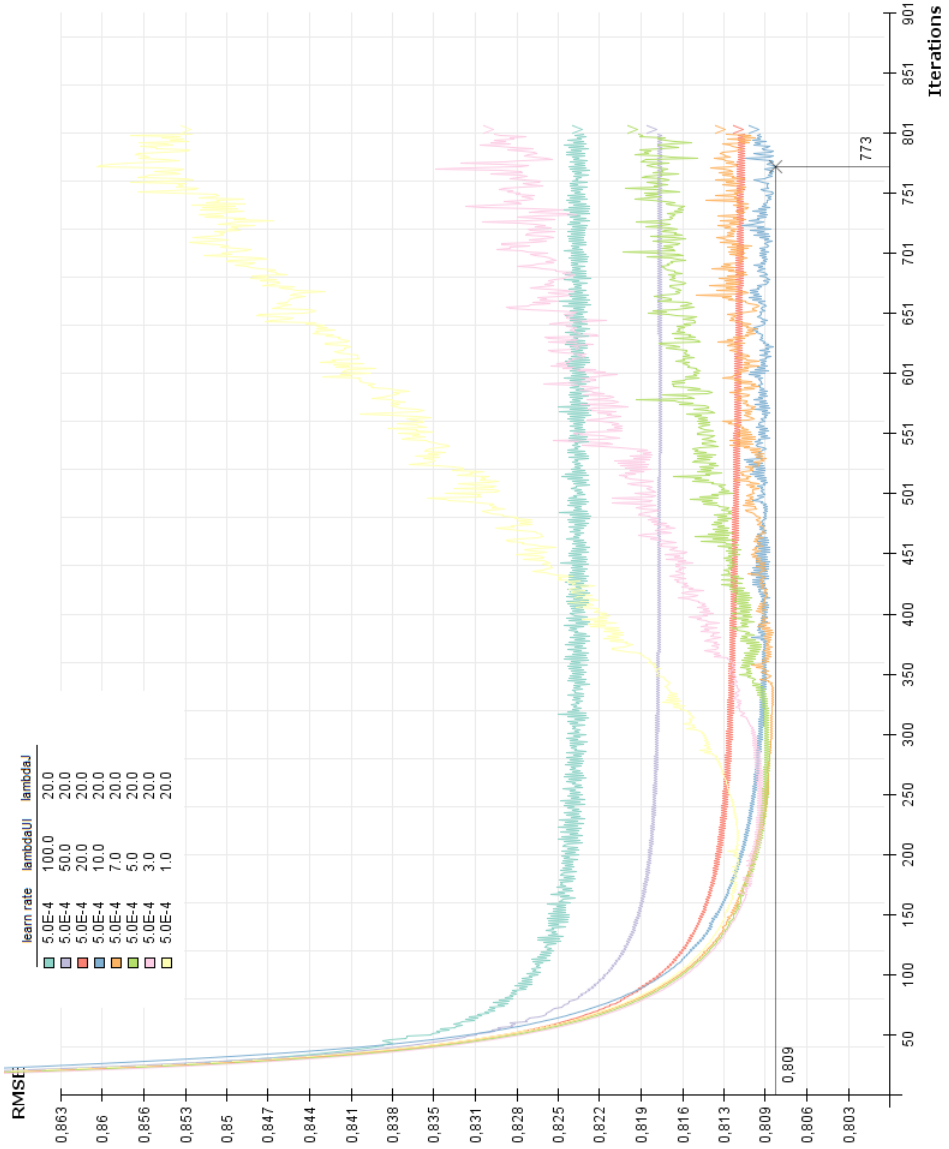


Figure 8.15: Validation error of SVD++ on Douban* with fixed $\lambda_U = \lambda_I = 10$. The model performs best for $\lambda_J = 20$.

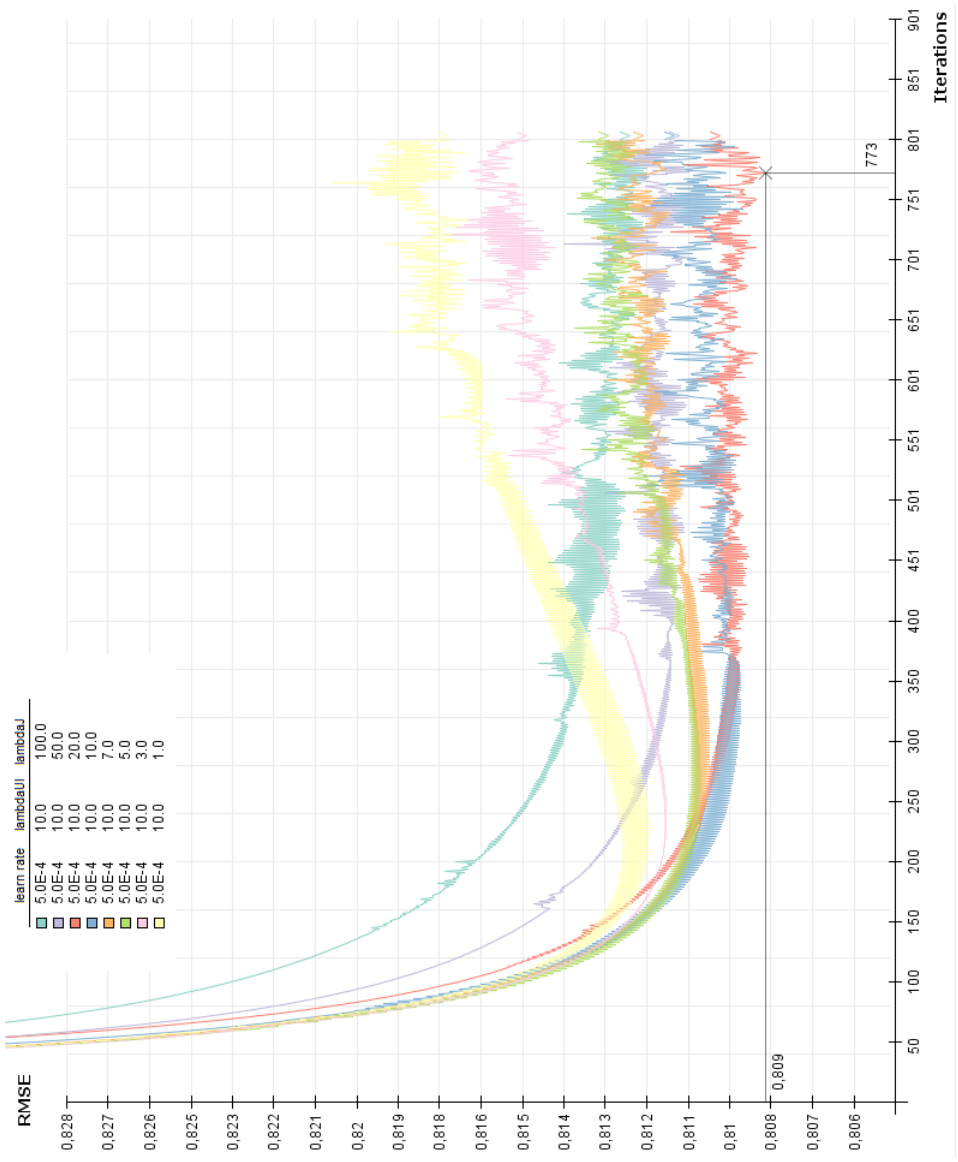


Figure 8.16: Validation error of SoRec on Douban*. The model performs best for $\lambda_U = \lambda_I = 3$, $\lambda_V = 1$, $\lambda_T = 10$ after 565 iterations.

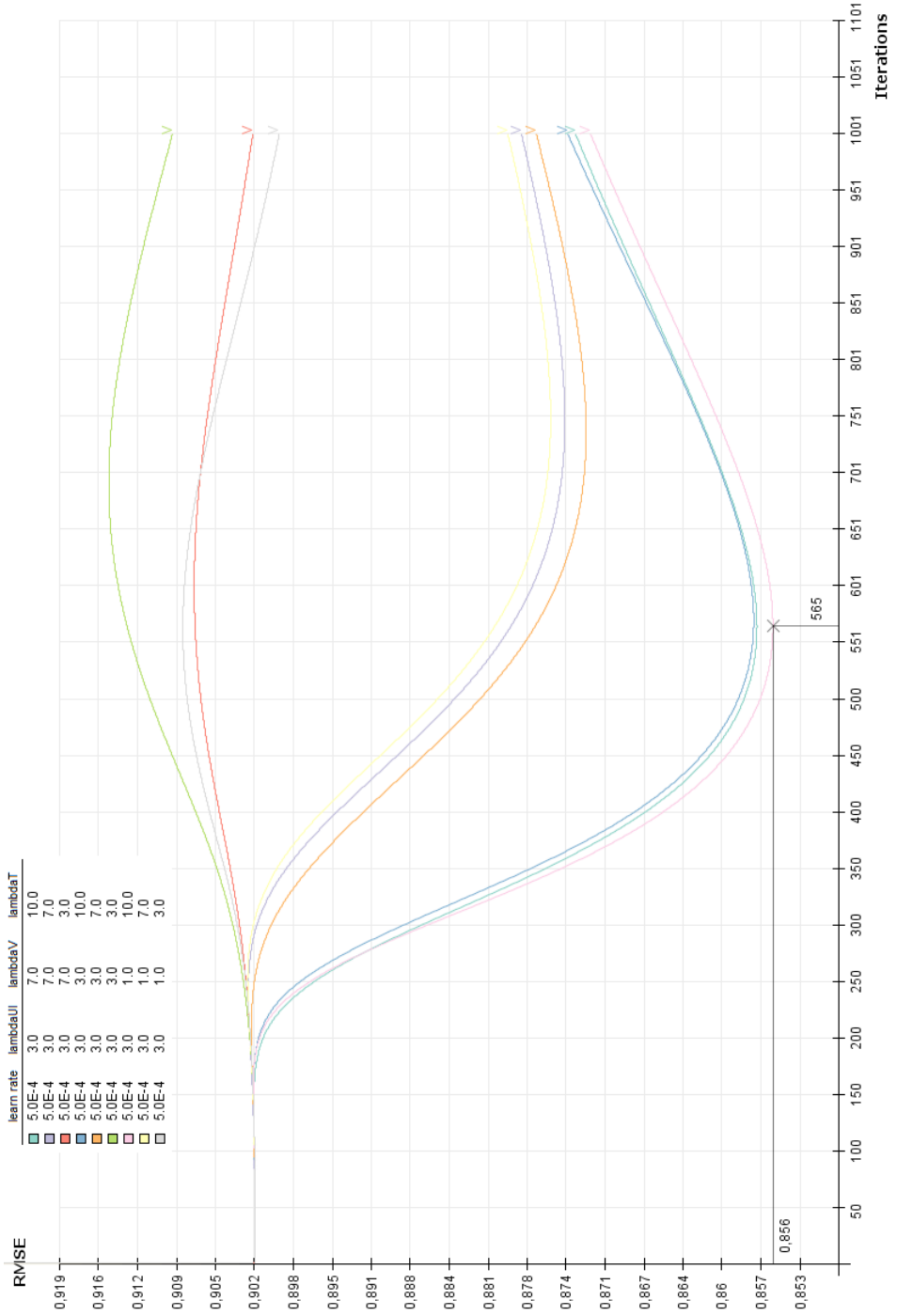


Figure 8.17: Validation error of RSTE on Douban* with fixed $\lambda_U = \lambda_I = 1$. The figure outlines the optimal setting for α is 0.2, found after 862 iterations.

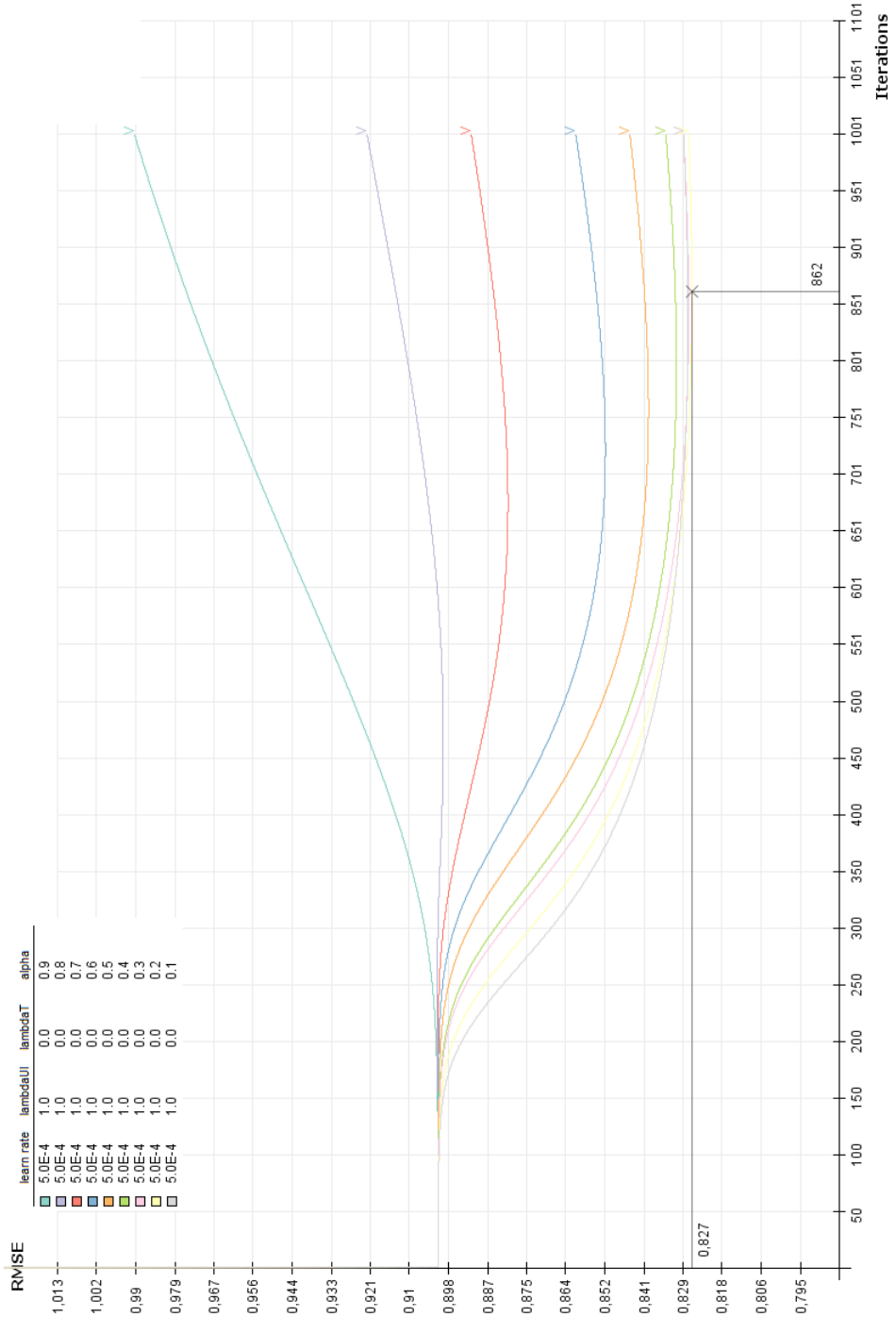


Figure 8.18: Validation error of RSTE on Douban* with fixed $\alpha = 0.2$. Within first 1000 iterations the best setting for $\lambda_U = \lambda_I$ is 1, however, the figure indicates that $\lambda_U = \lambda_I = 3$ would compensate up to 0.003 points in further iterations.

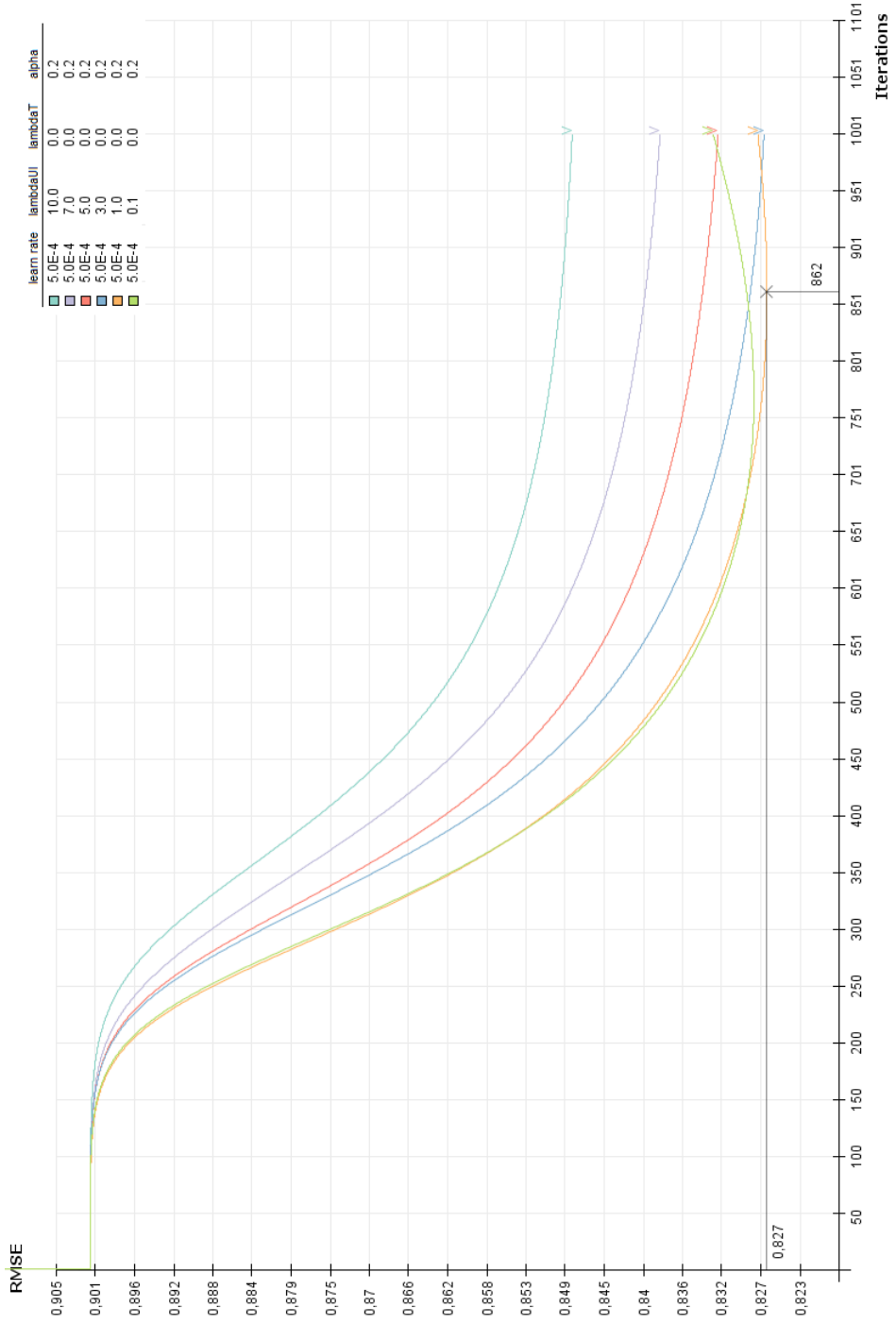
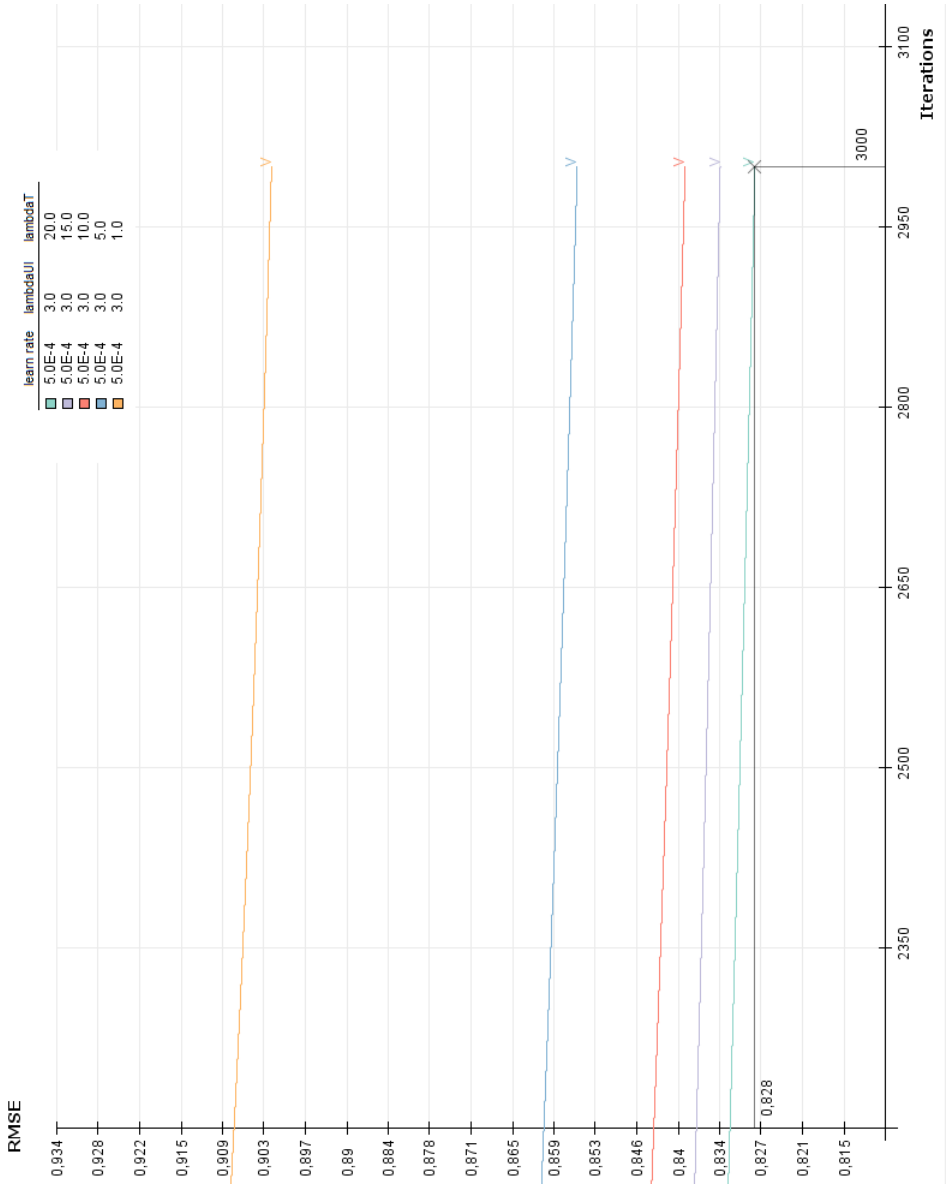


Figure 8.19: Validation error of SocialMF on Douban*. We stopped after 3000 iterations, where the best setting for λ_T has been 20. For finding the actual optimum an adaptive learn rate is necessary, since increasing the static learn rate leads to flickering.



8.6 Flixster

We also experimented with the Flixster data set, however, since the data set is very large, there could not be found optimal hyperparameters for *all* models yet. Remember the estimation in section 8.2 according to which a small hyperparameter search on Flixster might easily take up to 43 days. In order to compare to Epinions (cf. table 8.8) we also provide model runtimes for $k = 5$ on Flixster in table 8.12.

Table 8.12: Model runtimes on Flixster for $k = 5$.

		iteration over train set (ms)	**evaluation of valid. set (ms)**
k=5			
	PMF	4,536	405
	SVD++	617,515	2,625
	SoRec	28,8862	396
	RSTE	120,758	6,780
	SocialMF	15,212	398

8.7 Alternative Trust Weights

Throughout chapter 7 we presented several alternatives to weight trust in social-aware matrix factorization models. We implemented equal distribution of trust (s. 7.1) as used in the evaluation so far, HA-weights as proposed in section 7.2, the PageRank-based approach described in section 7.3, and also the weighting by rating similarity PPMCC measure presented in section 7.4.

Table 8.13 summarizes the errors of RSTE with different trust weights on Douban*. As expected PPMCC does not improve since most social connections simply drop out. Interestingly it even performs worse than PMF on Douban*, so that we suppose a new hyperparameter search is necessary as the structure of the trust matrix differs too much from the original. However, for HA-weights and PageRank the situation changes: besides improving the errors both weights also lead to earlier convergence of the models. With our proposed PageRank-weight there could be saved about 13% of the number of iterations, while at the same time improving the errors of RSTE by 0.005 points. In relation to the Netflix prize remember the announcement of 1,000,000 USD for improving the recommender by 10% from an RMSE of 0.9525 to 0.8572. Since on Douban* the RMSE

is already quite low with 0.827, the improvement of 0.6% should not be disregarded here. PageRank-weights as a preprocessing step to social-aware matrix factorization is also cheap, since we only need to calculate them once. For the full trust matrix of Douban the process does not even take a single minute. Also note that trusts are considered undirected and hyperparameters were reused from equal weights, so that there might be room left to further improve performance of PageRank-weights.

Table 8.13: Performances of trust weights in RSTE on Douban*.

		valid. error	**test error**	**learn rate**	**iterations**
k = 5					
	Equal	0.827	0.832	0.0005	862
	PPMCC	0.900	0.901	0.0005	550
	HA	0.826	0.829	0.0005	840
	PageRank	0.822	0.827	0.0005	752

We further experimented with alternative trust weights for SocialMF on Douban*, however, there could not be gathered results for all weights without a new hyperparameter search. Also the static learn rate again thwarts the evaluation, so that in table 8.14 we unfortunately have to provide gaps for PPMCC- and PageRank-weights.

Table 8.14: Performances of trust weights in SocialMF on Douban*.

		valid. error	**test error**	**learn rate**	**iterations**
k = 5					
	Equal	0.828	0.828	0.0005	3000
	PPMCC	?	?	0.0005	?
	HA	0.825	0.831	0.0005	3000
	PageRank	?	?	0.0005	?

8.8 Cold start users

A critical subtask of rating prediction is predicting ratings for users, who did not rate any items at all. For these *cold start users* it would be nice to have accurate predictions, too, so that we formulate performance of rating predictors on cold start users as a further criterion of robustness.

To measure the performance on cold start users we modify the evaluation protocol in that we do not randomly sample the test set among all ratings, but we put *all* ratings of randomly chosen users into the test set, such that we again receive about 50,000 ratings in total. Next we train the models with the already found hyperparameters and compute the RMSE against the random cold start user test set.

Table 8.15 shows the resulting errors on the Epinions data set. Again plain PMF is last in line, followed by SVD++ which strongly degenerates since cold start users do not provide a rating history. Therefore SVD++ cannot infer knowledge about item-item interactions from the additional matrix J in the prediction. RSTE is most robust towards cold start users on the Epinions set with only losing 0.042 points compared to its validation error — and improves over the test error of SVD++ by remarkable 4.7%. Also SoRec and SocialMF perform still well compared to their starting levels.

Table 8.15: Cold start performances on Epinions for $k = 5$.

		valid. error	**test error**	**learn rate**	**iterations**
k=5					
	PMF	1.139	1.232	0.005	132
	SVD++	1.097	1.206	0.0005	317
	SoRec	1.114	1.159	0.0005	601
	RSTE	1.107	1.149	0.0005	905
	SocialMF	1.126	1.182	0.0005	1630

For Douban* there shows up a similar picture, where again SVD++ degenerates most compared to its validation error. Furthermore findings about social-aware models are confirmed by the results of cold start user evaluation on Douban*, as the greater data set again balances out the errors of RSTE and SocialMF. SoRec, however, is most stable compared to its original random test set error by only losing 0.004 points on the new cold start user test set.

Table 8.16: Cold start performances on Douban* for $k = 5$.

		valid. error	test error	learn rate	iterations
k=5					
	PMF	0.851	0.907	0.005	720
	SVD++	0.807	0.899	0.0005	766
	SoRec	0.858	0.861	0.0005	572
	RSTE	0.827	0.838	0.0005	866
	SocialMF	0.826	0.841	0.0005	3000

8.9 Bias Terms

Throughout this section we would like to investigate the impact of bias terms in matrix factorization models, as presented in section 4.5. For the evaluation we switch back to the original variant of the test set which does not only concentrate on cold start users, but again we rely on hyperparameters as given in tables 8.3 and 8.10.

Table 8.17 provides the results on Epinions. We checked performance of pure bias terms by setting $k = 0$ in PMF, which surprisingly predominates all proposed models in terms of RMSE.

Table 8.17: Model performances with bias terms on Epinions.

		valid. error	test error	learn rate	iterations
k = 0					
	PMF	1.046	1.069	0.005	54
k = 5					
	PMF	1.047	1.073	0.005	56
	SVD++	1.044	1.069	0.0005	269
	SoRec	1.046	1.074	0.0005	312
	RSTE	1.049	1.070	0.0005	264
	SocialMF	1.043	1.072	0.0005	395

We suspect the outstanding performance of pure bias terms also compared to models without bias terms is reasoned by little variance in rating profiles of single users and items, so that bias terms can easily capture the rating behaviour. Furthermore all data sets show the tendency of users to more likely rate in case of positive experience with an item, which is

reflected in table 8.18. For completeness, we also provide numbers for full Douban and Filxster data sets.

Table 8.18: User and item rating profiles of the data sets.

	douban	**douban***	**epinions**	**flixster**
user rating profiles				
mean	4.0	4.0	4.0	3.7
st.dev.	0.76	0.69	0.70	0.52
item rating profiles				
mean	3.7	3.9	4.1	3.4
st.dev.	0.52	0.45	0.31	0.72
total rating profiles				
mean	3.8	4.0	4.0	3.6
st.dev.	0.91	0.90	1.21	1.09

Among all data sets the mean rating per user and item is higher than the average rating 3 (remember the scale is 1 to 5). Also the standard deviations in user ratings are quite equal among the data sets — on Flixster user ratings are most balanced with a standard deviation of only 0.52. The lower user rating mean of 3.7 here further suggests that the proportion of discriminating users is higher in Flixster. For Epinions and Douban* there can be noticed very similar rating means per user, per item and in total, and also the standard deviations per user ratings are quite equal. The total standard deviation 1.21 of Epinions is higher than 0.90 of Douban*, so that we would expect Epinions to be a good target for improving accuracy with factorization models, rather than Douban*.

Table 8.19: Model performances with bias terms on Douban*.

		valid. error	**test error**	**learn rate**	**iterations**
k = 0					
	PMF	0.838	0.841	0.005	57
k = 5					
	PMF	0.839	0.851	0.005	58
	SVD++	0.766	0.771	0.0005	394
	SoRec	0.768	0.774	0.0005	288
	RSTE	0.763	0.768	0.0005	447
	SocialMF	0.763	0.769	0.0005	438

However, when looking at results on Epinions in table 8.17 we can not observe significant improvements over pure bias terms from any of the factorization models. In comparison, when regarding results of table 8.19 which presents results on Douban*, we see that extended factorization models with bias terms here outperform pure bias terms and PMF, although in Douban* there is less standard deviation in total than in Epinions. But since bias terms capture the bias per user and item, the total standard deviation is rather worthless to look at - instead we should check the standard deviation of user and item rating profiles, which suggests that Epinions is less ambiguous than Douban*. From this point of view there is more room for improvement by extended factorization models on Douban* than on Epinions, which is also confirmed by the findinds in table 8.19.

9 Conclusions

In the retrospection we reviewed and categorized early approaches in collaborative filtering, which were found to be outperformed by matrix factorization models as a result of the Netflix prize. We worked out similarities of available social-aware factroization models and derived a generic model, which has been shown to be adaptable to special models. By appropriate caching we could improve runtime complexities. We further transposed the social-aware models for the incorporation of presented trust weights. Moreover we proposed PageRank-weights as an alternative trust metric.

Throughout the evaluation we found social-aware models (especially RSTE) tend to be more robust on smaller data sets than SVD++, and clearly outperform SVD++ on the cold start user problem. Although not revolutionizing the accuracy of social-aware factorization models, alternative trust weights such as e.g. PageRank-weights did at least slightly improve the RMSE and reduced the number of training iterations by up to 13% for the RSTE model.

Lastly bias terms were found to predominate performance of matrix factorization models, such that for data sets with lower variance in user and item rating profiles factorization models even deteriorated prediction with pure bias terms. However, the situation changes for larger data sets with higher variance such as Douban*, where social-aware factorization models and SVD++ could justify there *raisons d'être* and take the lead again.

Future Work

Throughout this work we focused on undirected trust information, however, tipping trust weights could also be improved by the directed *follower* style (e.g. Twitter[1]), where opinion leaders are already given apriori. Alternatively, for undirected data sets it would be nice to have additional

[1] `http://twitter.com`

information about friendship (e.g. messaging frequencies), on which trust weights could be further refined.

For model training with full gradient descent we also experienced that static learn rates may become a problem depending on the size of the data set, so that we propose to rely on the stochastic version of the algorithm and/or make use of a learn rate which adapts to the number of iterations.

Also we suffered expensive searches for hyperparameters, where for bigger data sets we could not even accomplish the task in a realiable way. In this regard, [31] proposed adaptive regularization, where hyperparameters can be learned in parallel. Adaptive regularization could be the key to properly tackle the presented larger data sets with social-aware matrix factorization models, since large-scale hyperparameter search would drop out.

Within our proposed generic social-aware model there can also be formulated and evaluated alternative assumptions on the influence of trust via the similarity matrix M.

A List of Abbreviations

ALS	Alternating Least Squares
CF	Collaborative Filtering
MAE	Mean Absolute Error
MF	Matrix Factorization (cf. section 3.1)
PMF	Probabilistic Matrix Factorization (cf. section 4.4)
PPMCC	Pearson Product Moment Correlation Coefficient
(R)STE	(Recommendation with) Social Trust Ensemble (cf. section 5.2)
RMSE	Root mean square error
(S)GD	(Stochastic) Gradient Descent
SocialMF	Social Matrix Factorization (cf. section 5.3)
SoRec	Social Recommendation (cf. section 5.1)
SVD++	State-of-the-Art extension to PMF (cf. section 4.6)

B List of Notations

n	Total number of users
m	Total number of items
k	Number of latent features
R	Sparse rating matrix of observations $\in \mathbb{R}^{n \times m}$
$\bar{R}_{u,\cdot}$	Mean rating of user u
$\bar{R}_{\cdot,i}$	Mean rating of item i
X	Binary matrix $\in \mathbb{R}^{n \times m}$ stating whether $(u, i) \in R$
U	Latent user feature matrix $\in \mathbb{R}^{n \times k}$
I	Latent item feature matrix $\in \mathbb{R}^{m \times k}$
T	Sparse trust matrix $\in \mathbb{R}^{n \times n}$ containing weigths of trust between two users in the network
Y	Binary matrix $\in \mathbb{R}^{n \times n}$ stating whether $(u, v) \in T$
$N(u)$	Set of users who are trusted by u, i.e. $\forall v \in N(u) : (u, v) \in T$
$\mathcal{B}(u)$	Set of users who trust u, i.e. $\forall v \in \mathcal{B}(u) : (v, u) \in T$
ω^R	Global average rating
ω^U	Vector $\in \mathbb{R}^n$ containing average ratings per user
ω^I	Vector $\in \mathbb{R}^m$ containing average ratings per item
λ	Hyperparameter for regularization
I^u	Set of items rated by user u
U^i	Set of users who rated item i u
$\mathcal{R}(u, i)$	Predicted rating for item i of user u
$\mathcal{P}(u, i)$	Prediction error $R_{u,i} - \mathcal{R}(u, i)$
$\mathcal{L}(...)$	Loss function
$\|\| \cdot \|\|_F$	Frobenius norm for matrices
$\|\| \cdot \|\|$	Euclidean norm for vectors

C Source code

The implementation used for the evaluation in chapter 8 can be found on the attached storage device, together with a digital copy of this work. In order to build the code make sure you have installed Java (at least JDK 6[1]), as well as the build management tool Apache Maven 2.2.1[2]. The source code can be compiled from within the project root folder by executing

```
mvn clean package
```

After a successful build the packages can be found inside the *target*-folder.

Prebuilt packages are also supplied on the storage device, however, since these were compiled on Windows 7 (64-bit), if they are executed in an alternative environment it is recommended to rebuild the sources there.

The build consists of three zip-files

- *jrec-ratingpred-data.zip*: contains the data sets
- *jrec-ratingpred-windows.zip*: contains executable batch scripts for Windows
- *jrec-ratingpred-linux.zip*: contains executable shell scripts for Linux

There exist predefined executables for the implemented models `PMF`, `SVD++`, `SoRec`, `RSTE`, `SocialMF`, as well as the memory-based `PPMCC` approach — one for each data set `Epinions`, `Douban` and `Flixster`.

Note that some of the parameters are shared among the models, e.g. the hyperparameter `alpha` is used as `lambdaV` in SoRec. Details on parameter usage are displayed when simply executing e.g. `run_ratingpred.bat` on Windows, so that the manual of the command line interface is presented.

Also note that hyperparameters and the learn rate can deal with comma-separated values in order to batch evaluate different model settings.

[1] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
[2] `http://http://maven.apache.org`

D Acknowledgements

First and foremost I want to thank Assistant Prof. Dr. Steffen Rendle, who opened my mind for recommender systems in general, and supported this thesis by valuable suggestions and way-paving ideas. Also I want to thank Dr. Christoph Freudenthaler, not only for precise code reviews, but also his excellent soccer skills, which enriched our team. Moreover I want to thank David Schoch, who accomodated me in his office, explained to me the undreamed functionality of a professional coffee brewer, and always lend a willing ear for discussions. I also thank Matthias Fratz for several discussions and who showed me how to clean a coffee brewer with undreamed functionality. Last but not least I thank Prof. Dr. Daniel A. Keim as second reviewer of this work, and his commitment to mentorship throughout my studies.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, -06 2005.

[2] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, March 1997.

[3] J. Bennett, C. Elkan, B. Liu, P. Smyth, and D. Tikk. Kdd cup and workshop 2007. *ACM SIGKDD Explorations Newsletter*, 9(2):51–52, 2007.

[4] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[5] Daniel Billsus and Michael J. Pazzani. A personal news agent that talks, learns and explains. In *Proceedings of the third annual conference on Autonomous Agents*, AGENTS '99, pages 268–275, New York, NY, USA, 1999. ACM.

[6] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[8] Ward Cheney and David Kincaid. Linear algebra: Theory and applications. *The Australian Mathematical Society*, page 654, 2009.

[9] David Maxwell Chickering, David Heckerman, and Christopher Meek. A bayesian approach to learning bayesian networks with local structure. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, UAI'97, pages 80–89, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[10] William W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1*, AAAI'96, pages 709–716. AAAI Press, 1996.

[11] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[12] Hwai-Hui Fu, Dennis KJ Lin, and Hsien-Tang Tsai. Damping factor in google page ranking. *Applied Stochastic Models in Business and Industry*, 22(5-6):431–444, 2006.

[13] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, July 2001.

[14] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 194–201, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[15] Netflix Inc. *Netflix Prize*, 2006-2009. URL: `http://netflixprize.com`.

[16] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 135–142, New York, NY, USA, 2010. ACM.

[17] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 79–86, New York, NY, USA, 2010. ACM.

[18] Y. Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[19] Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[20] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[21] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[22] Ken Lang. Newsweeder: Learning to filter netnews. In *in Proceedings of the 12th International Machine Learning Conference (ML95*, 1995.

[23] Hao Ma, Irwin King, and Michael R. Lyu. Learning to recommend with social trust ensemble. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 203–210, New York, NY, USA, 2009. ACM.

[24] Hao Ma, Haixuan Yang, Michael R. Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 931–940, New York, NY, USA, 2008. ACM.

[25] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 287–296, New York, NY, USA, 2011. ACM.

[26] Raymond J. Mooney, Paul N. Bennett, and Loriene Roy. Book recommending using text categorization with extracted information. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)"-REC-WKSHP98, year="1998*, pages 70–74, Madison, WI, 1998.

[27] John O'Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174. ACM, 2005.

[28] Michael Pazzani and Daniel Billsus. Learning and revising user profiles: The identification ofinteresting web sites. *Mach. Learn.*, 27(3):313–331, June 1997.

[29] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 473–480, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[30] I. Pilászy and D. Tikk. Computational complexity reduction for factorization-based collaborative filtering algorithms. *E-Commerce and Web Technologies*, pages 229–239, 2009.

[31] Steffen Rendle. Learning recommender systems with adaptive regularization. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 133–142, New York, NY, USA, 2012. ACM.

[32] Jasson DM Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[33] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

[34] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust management for the semantic web. *The Semantic Web-ISWC 2003*, pages 351–368, 2003.

[35] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70. ACM, 2002.

[36] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465 – 471, 1978.

[37] H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[38] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. *Advances in neural information processing systems*, 20:1257–1264, 2008.

[39] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating 'word of mouth'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[40] N. Srebro, J.D.M. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. *Advances in neural information processing systems*, 17(5):1329–1336, 2005.

[41] Iraklis Varlamis, Magdalini Eirinaki, and Malamati Louta. A study on social network metrics and their application in trust networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, pages 168–175. IEEE, 2010.

[42] Albert Au Yeung. Matrix factorization: A simple tutorial and implementation in python. `http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python`, September 2010.

[43] K. Yu, X. Xu, J. Tao, M. Ester, and H.-P. Kriegel. Instance selection techniques for memory-based collaborative filtering. In *Proceedings of Second SIAM International Conference on Data Mining (SDM?02)*, 2002.

[44] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 116–, New York, NY, USA, 2004. ACM.

[45] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. *Algorithmic Aspects in Information and Management*, pages 337–348, 2008.