AALTO UNIVERSITY
SCHOOL OF SCIENCE
Department of Information and Computer Science
Degree Programme of Computer Science and Engineering

Jussa Klapuri

Collaborative Filtering Methods on a Very Sparse Reddit Recommendation Dataset

Master's thesis

Espoo, 28th January, 2013

Supervisor: Prof. Erkki Oja

Instructor: Ilari Nieminen, M.Sc.(Tech.)

Aalto University
School of Science
and Technology

AALTO UNIVERSITY SCHOOL OF SCIENCE Department of Information and Computer Science Degree Programme of Computer Science and Engineering	ABSTRACT OF MASTER'S THESIS

Author:	Jussa Klapuri
Title:	Collaborative Filtering Methods on a Very Sparse Reddit Recommendation Dataset

Number of pages:	vii + 48	Date:	28th January, 2013	Language:	English
Professorship:	Computational Science	Code:			T-61
Supervisor:	Prof. Erkki Oja				
Instructor:	Ilari Nieminen, M.Sc.(Tech.)				

Abstract:

The research question in this thesis concerns how accurately it is possible to estimate users' future votes based on their voting history when votes can only be for or against (upvotes or downvotes). This estimation is done on a large and very sparse dataset of over 23 million votes gathered from the website Reddit, which is a very popular social news and entertainment website where users may vote content up or down. Since over 84% of all the votes are upvotes, downvote estimation is made even more difficult. Similar to the Netflix Prize, collaborative filtering (CF) methods can be used to approach this problem.

The two main approaches in CF are neighborhood models and latent factor models. Models using both approaches are implemented and tested in this thesis. Neighborhood approach is implemented by using a k-nearest-neighbor (k-NN) classifier with 3 different types of feature vectors and latent factor models are represented by the classical Singular Value Decomposition (SVD) and more advanced Variational Bayesian Principal Component Analysis (VBPCA). While both SVD and VBPCA are implemented with regularization, VBPCA also uses Bayesian inference methods by adding the noise term into the model and introducing prior distributions over the model parameters.

For the experiments, the full Reddit dataset is preprocessed into 3 different sized datasets (k-cores), where the biggest dataset represents the full dataset after removing some noise and outliers and the smallest dataset represents the core of the data, i.e., the most active users and the most voted links. The middle dataset has some properties from both the small and the big dataset. All models are tested for each dataset and the results are measured using different metrics, most notably RMSE. VBPCA and link-based k-NN model are shown to perform best on all datasets. While VBPCA is better in terms of downvote estimation for the small and the middle datasets, link-based k-NN performs best on the big dataset.

Keywords: Recommender Systems, Collaborative Filtering, Sparse Dataset, Singular Value Decomposition, Variational Bayesian PCA, k-Nearest Neighbors, Stochastic Gradient Descent, Reddit, Netflix

Aalto-yliopisto
Teknillinen korkeakoulu

AALTO-YLIOPISTO Perustieteiden korkeakoulu Tietotekniikan tutkinto-ohjelma			DIPLOMITYÖN TIIVISTELMÄ
Tekijä:	Jussa Klapuri		
Työn nimi:	Yhteistoiminnalliset Suodattamismenetelmät Hyvin Harvalle Reddit-datalle		
Sivumäärä:	vii+ 48	Päiväys: 28. tammikuuta, 2013	Kieli: englanti
Professuuri:	Laskennallinen Tiede	Professuurikoodi:	T-61
Työn valvoja:	Prof. Erkki Oja		
Työn ohjaaja:	DI. Ilari Nieminen		

Tiivistelmä:

Diplomityön tutkimuskysymys on, kuinka tarkasti on mahdollista estimoida käyttäjien tulevaa äänestyskäyttäytymistä heidän äänestyshistoriansa perusteella, kun äänet ovat joko puolesta tai vastaan. Estimointi suoritetaan laajalle ja hyvin harvalle data-aineistolle, joka koostuu 23 miljoonasta äänestä Reddit-sivustolta. Koska yli 84% äänistä on puolesta, vastaan olevien äänien estimointi on vielä vaikeampaa. Samaten kuin Netflix prize -kilpailussa, yhteistoiminnallisia suodattamismenetelmiä (collaborative filtering, CF) voi käyttää tämän ongelman ratkaisuun.

Kaksi päälähestymistapaa CF-menetelmillä ovat naapurusto- ja latenttimuuttujamallit, joista kummastakin toteutetaan ja testataan menetelmiä. Naapurustomenetelmä toteutetaan k:n lähimmän naapurin (k-NN) menetelmän avulla ja aliavaruusmenetelmistä käytetään esimerkkinä pääakselihajotelmaa (SVD) sekä variaatioapproksimoitua pääkomponenttianalyysia (VBPCA). Vaikka molemmat SVD ja VBPCA on toteutettu regularisaatiolla, VBPCA käyttää myös bayesiläistä päättelyä lisäämällä kohinatermin malliin sekä mallintamalla priorijakaumat mallin parametreille.

Kokeita varten koko Reddit-aineisto esikäsitellään kolmeksi erikokoiseksi aineistoksi, joista isoin vastaa melkein koko Reddit-aineistoa kun datasta on ensin poistettu kohinaa ja poikkeavia havaintoja. Pienin aineisto vastaa koko aineiston ydintä, siis kaikkein aktiivisimpiä käyttäjiä ja äänestetyimpiä linkkejä. Keskimmäinen aineisto sisältää ominaisuuksia sekä pienestä että isosta aineistosta. Kaikki menetelmät ajetaan kaikille aineistoille ja suorituskykyä mitataan eri mittareilla, pääasiassa keskineliövirheen neliöjuurella (RMSE).

Kokeissa havaitaan, että VBPCA ja linkkipohjainen k-NN osoittautuvat testattavista menetelmistä parhaiksi kaikille aineistoille. Vaikka VBPCA onkin parempi kahdelle pienimmälle aineistolle, linkkipohjainen k-NN on parempi suurimmalle aineistolle.

Asiasanat:	Suositusjärjestelmät, Yhteistoiminnallinen Suodattaminen, Harva data, Pääakselihajotelma, Pääkomponenttianalyysi

Preface and Acknowledgements

I wish to thank the Department of Information and Computer Science at Aalto University for hiring me as a summer student twice through my studies and especially the VirtualCoach research project for funding this thesis. I am particularly grateful to Krista Lagus and my instructor Ilari Nieminen for getting me involved in this health and wellbeing related project in the first place and thanks to the whole VirtualCoach research and partner team, especially to Lassi Haaranen and Antti Heikkilä for decreasing my workload considerably to be able to focus on completing this thesis.

Special thanks to Prof. Erkki Oja for supervision and valuable comments and Tapani Raiko for great ideas and for lending me the Recommender Systems Handbook that became the integral reference for this thesis.

Thanks to Taneli Riitaoja and Artturi Tilanterä for providing feedback and comments on different parts of the text. Finally, thanks to my family and close ones for supporting me through this thesis project.

Espoo, 28th January, 2013
Jussa Klapuri

Contents

List of Symbols

$\boldsymbol{M}$	A matrix
$\boldsymbol{x}$	A vector
x	A scalar
N	Number of rows (observations) in a dataset (indexed by i)
M	Number of columns (variables) in a dataset (indexed by j)
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution
μ	Mean
σ^2	Variance
Σ	Covariance matrix
$\mathcal{N}_i(u)$	Set of closest neighbors of user u having voted link i
$\mathcal{U}$	Set of users
$\mathcal{R}$	Set of votes
K	Number of folds in K-fold cross validation
$\mathcal{K}$	Set of known votes
$\mathcal{I}$	Set of links
$\mathcal{S}$	Set of possible values for votes
u, v	Single votes
i, j	Single links
d	Dimension
k	The number of nearest neighbors in k-NN or the number of components in SVD
r_{ui}	Value of vote by user u to item i
$\hat{r}_{ui}$	Predicted value of vote by user u to link i
$\bar{r}_u$	Average rating of user u
$\bar{r}_i$	Average rating of link i
e_{ui}	Error term for vote to link i given by user u
β	Weight in linear regression
ϵ	Gaussian noise term
P	Probability
γ	Learning rate
λ	Regularization parameter
$C(\cdot)$	Cost function
$\boldsymbol{\theta}$	Parameter vector
$\boldsymbol{\xi}$	Hyperparameter vector

List of Abbreviations

ARD	Automatic Relevance Determination
CF	Collaborative Filtering
EM	Expectation Maximization
i.i.d	independent and identically distributed (random variables)
k-NN	k Nearest Neighbors
MAR	Missing At Random
ML	Maximum Likelihood
MLRM	Multiple Linear Regression Model
PCA	Principal Component Analysis
PDF	Probability Density Function
PPCA	Probabilistic Principal Component Analysis
RMSE	Root Mean Square Error
RS	Recommender System
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
VBPCA	Variational Bayesian PCA

Chapter 1

Introduction

1.1 Problem Setting

Recommender systems are software tools and techniques providing suggestions for items or objects that are assumed to be useful to the user. These suggestions can relate to different decision-making processes, such as which books might be interesting, which songs you might like, or people you may know in a social network. (Ricci et al., 2011, p. 1) In this thesis, the particular interest of an RS is that of reddit.com (Reddit), which is a type of online community where *users* can vote *links* either up or down, i.e. *upvote* or *downvote*. Reddit currently has a global Alexa rank of 134 and 65 in US (Alexa), which measures the combination of average daily visitors and page views over a period of three months, making it one of the most popular sites on the internet as of early 2013.

The front page of Reddit shows 25 popular stories or links for users, which can be user-tailored. This ranking algorithm mainly depends on the popularity, i.e., the votes given to the story, as well as the time frame of votes given (Salihefendic, 2010). This thesis does not attempt to improve nor analyze this ranking any further, rather, the research question of this thesis is:

> *Given a priori knowledge of a user's voting history, will the user like or dislike some specific content*?

This approach means that when the link is given, for example highly ranked link on the front page of Reddit, will the user upvote or downvote that link. Some practical applications of this kind of mechanism could be to highlight the links a particular user might like or to hide links that are assumed to be downvoted by the user, though hiding bad links would cause even less links to be explicitly downvoted by the users, leading to more difficulty in estimation (more details in Section 5.1). By using a large sparse dataset containing millions of *votes* from thousands of users, this thesis attempts to estimate the expected vote given by a user for a particular link by using different methods and comparing their performance.

This kind of problem can generally be approached efficiently by using *collaborative filtering* (CF) methods. In short, collaborative filtering methods produce user specific recommendations of links based on voting patterns without any need of exogenous information about either users or links (Ricci et al., 2011, p. 145). Unlike many

other theses using similar models (see e.g. Vatanen, 2012; Noeva, 2012)for multiple datasets, the main focus in this thesis is to apply CF techniques to the little-known Reddit recommendation dataset (King, 2010). These techniques have gained a lot more popularity after the winner of the recently completed $1 million Netflix Prize competition (Netflix, 2009; Bell et al., 2007) used CF techniques in a central role.

The Netflix Prize competition was originally launched in 2006. At the time, Netflix was mostly known as a DVD rental company, while nowadays it is the world's leading internet subscription service for films and TV programmes (Netflix). The company decided to launch the competition in order to improve their algorithm in recommending films based on personal preferences. The provided training dataset contained 100,480,507 ratings from 480,189 users on 17,770 movies resulting in a sparse matrix where only 1.2% of elements contain data (Netflix, 2009). To compare, the Reddit dataset used in this thesis contains 23,091,688 votes from 43,976 users over 3,436,063 links resulting in a sparse matrix containing only 0.015% of the data in the full matrix. A comparison of properties between the popular Netflix dataset and the less known Reddit dataset is included in Table 1.1.

	Users	Items/Links	Ratings/Votes	Density
Netflix dataset	480,189	17,770	100,480,507	0.0118
Reddit dataset	43,976	3,436,063	23,079,454	0.000153

Table 1.1: Comparison between Netflix and Reddit dataset properties. Density represents the ratio of values observed in the full user-item matrix.

Due to sparsity reasons, the dataset was preprocessed in three different ways to create three different sized datasets such that the biggest dataset contains almost all of the data and is extremely sparse whereas the smallest dataset contains only about 10% of the votes in the original dataset and is closer to the sparsity level of the Netflix dataset. The size of the third dataset, 6.6 million votes, lays in between the two datasets and has some properties from both datasets. See Section 5.1 for more details on the preprocessed datasets for which all the methods were run. The missingness mechanism is assumed to be *missing at random* (MAR) and all of the methods presented in this thesis have the ability to deal with highly sparse data and to generalize to new data to some extent. The general dichotomy of CF approaches are *neighborhood-based models* and *latent factor models*. Both kinds of approaches are implemented and tested on all datasets with some varying parameters.

1.2 Contribution of the Thesis and Related Work

During the writing of this thesis, the only other work publicly found reporting a similar approach to the dataset was an exercise work for the "CS 229: Machine Learning" course at Stanford University (Poon et al., 2011). They documented their preprocessing approach and its parameters well enough for others to be able to produce a similar dataset. They also gave their invaluable results, which helped to compare the results in this thesis against theirs. However, during the writing process it became

evident that the evaluation measure they used for analyzing the results (*RMSE*, see Section 4.1) was probably not the single best measure for the spectrum of this thesis. Also, since the strict preprocessing left only about 10% of the votes in the original full Reddit dataset, it seemed reasonable to delve more deeply into the larger dataset. That is the main reason for splitting the original dataset into 3 smaller datasets.

This thesis was written in a research project called *VirtualCoach* (Lagus) and the dataset to be used was originally supposed to be provided by the project prototype website called *pathsofwellbeing.com*. The website has produced datasets in the form of stress questionnaire and nursing survey. Some of the users were partially active in a Questions & Answers forum that is somewhat similar to Stack Exchange (StackExchange). However, the number of users and the resulting dataset describing their behavior was clearly too small that any real scientific analysis could have been applied to it. Luckily, my instructor, Ilari Nieminen, discovered an enormous dataset of real world data from Reddit describing users' history on voting specific links up or down, based on whether the stories and discussions contained in the links are important (King, 2010).

1.3 Structure of the Thesis

This thesis is organized as follows. Chapter 2 gives an overview on recommender systems and collaborative filtering along with some basic terminology. All the methods used in the experiments section are described in Chapter 3. This includes the naive and baseline predictors, Singular Value Decomposition (SVD) and k-Nearest Neighbors (k-NN). Variational Bayesian Principal Component Analysis (VBPCA) is described in Section 3.4, which starts by defining Principal Component Analysis (PCA) and Probabilistic PCA (PPCA). Chapter 4 reviews the commonly used evaluation metrics for comparing the results of experiments, most notably the Root Mean Square Error (RMSE) and Receiver Operating Characteristic (ROC) curves. This chapter also presents the K-Fold Cross Validation used in determining the best number of components k for SVD and the number of neighbors k for k-NN. Though k is used as a parameter for both methods, they mean very different things and should not be confused. In Chapter 5, the full Reddit dataset is analyzed in detail and its pruning into three different sized datasets: small, mid and big, is described. The vote estimation task is then conducted for the three datasets and the results are analyzed at the end of the chapter. Chapter 6 concludes the thesis with discussion and some directions for future work.

Chapter 2

Recommender Systems

The study of recommender systems (RS) is relatively new compared to research of more classical information tools and techniques, such as search engines and databases. Recommender systems development began from a simple yet useful observation: individuals, or users, often rely on others to provide them with recommendations on making routine, daily decisions (Mahmood and Ricci, 2009; McSherry and Mironov, 2009). For example, it is common to rely on one's friends for recommending the next book to read or trust a film critic's movie review.

In order to replicate this behavior, the first RSs applied algorithms to leverage recommendations given by some community of users to the active user looking for suggestions. The recommendations were items that other similar users, i.e., users with similar tastes, had liked. This approach is termed *collaborative filtering* and its basic idea is that if the active user has agreed with some other users in the past, the active user will probably agree with the same users in the future and thus the recommendations coming from these other users should be relevant or interesting. (Ricci et al., 2011, p. 2)

When e-commerce Web sites started to develop, it soon became apparent that there was a pressing need for providing recommendations from filtering the whole range of alternatives. The exponential growth and variety of information available on the Web frequently overwhelmed users, leading them to make poorer decisions (Ricci et al., 2011, p. 2). The availability of choices started to decrease some users' well-being instead of providing benefits. It seemed that while choice is good, more choice is not always better for everyone. Thus, while choice has implications to freedom, autonomy and self-determination, excess choice may lead freedom to be regarded as a kind of "misery-inducing tyranny". (Schwartz, 2004) In recent years RSs have proved to be valuable in coping with the whole information overload problem. In the end, a recommender system solves this problem by pointing users towards new items that are not yet discovered, but may be relevant to their current task. (Ricci et al., 2011, p. 3)

From the service providers' point of view, there are various reasons why this technology is exploited. Probably the most important function for a commercial RS is to increase the number of items sold while selling more diverse items also attracts new users to the service. Also, an RS may increase user fidelity by providing better recommendations over time, leveraging the information acquired from the user in previous

interactions. (Ricci et al., 2011, p. 5)

It seems evident that there are several important reasons as to why service providers introduce RSs, but users may also want an RS if it will help them to support their tasks or reaching their goals. The classical reference paper in this field is by Herlocker et al. (2004), which defines eleven popular tasks that RSs may help implement. These tasks contain some main or core tasks that are generally associated with an RS, such as suggesting items that may be useful to a user, while other tasks are more related to "opportunistic" ways to exploit an RS, such as checking the importance of a Web page by checking its position in Google search results.

Recommendation systems are typically classified into six categories (Burke, 2007):

- content-based
- collaborative filtering
- demographic
- knowledge-based
- community-based
- hybrid recommender systems

Without delving deeper into these different RS types, the nature of the Reddit dataset practically rules out 5 classes of these, leaving only collaborative filtering based methods, which will be the major topic in this chapter. For example, using content-based methods in this application would require some external data, e.g. the actual contents of the links or messages voted. With this kind of information it would be possible to apply some text mining techniques to it, for example.

In general, recommender systems can rely on implicit or explicit feedback from the user. Both types of feedback can also be used in a single recommender system. Explicit feedback is the most convenient and means having the user explicitly rate links, whereas implicit feedback indirectly reflects user preferences. (Ricci et al., 2011, p.146) In the particular dataset used for this thesis, having certain implicit data would help the classification process enormously. For example, the dataset shows what user has voted for a link, if the user has indeed voted something, but for missing values there is no way of knowing whether a user has seen a link and decided not to vote, or simply has not seen a link at all. This is related to the missingness mechanism, which is assumed to be MAR (see Section 5.1).

,

2.1 Collaborative Filtering

Collaborative filtering (CF) methods produce user specific recommendations of links based on patterns of votes without any need of exogenous information about either users or links (Ricci et al., 2011, p. 145). The research in the field of collaborative filtering has advanced a lot after the Netflix Prize (Netflix, 2009) was announced in October 2006. This was the first time the research community was granted access to

a large-scale, industrial strength dataset of 100 million movie ratings. This attracted thousands of new people to this field, mainly scientists, engineers, students and enthusiasts. Because of the competition aspect, rapid development was encouraged where participants built on each generation of techniques in order to improve prediction accuracy.

CF systems need two fundamentally different entities in order to establish recommendations: items and users. While the term "item" is the general term in the literature, the rest of the thesis will mostly use the term *link* which represents the nature of the Reddit dataset better. With users and links, conventional techniques model the data as a sparse user-link matrix, which has a row for each user and a column for each link. The nonzero elements in this matrix are the votes.

The two main techniques of CF to relate users and links are *the neighborhood approach* and *latent factor models*. The neighborhood methods are based on finding similar neighbors to either links or users and computing the prediction based on these neighbors' votes, for example, finding k nearest neighbors and choosing the majority vote. Latent factor models approach this problem in a different way by attempting to explain the observed ratings by uncovering some latent features from the data. These models include neural networks, Latent Dirichlet Allocation, probabilistic Latent Semantic Analysis and SVD-based models (Ricci et al., 2011, p. 151). In this thesis, the k-nearest-neighbors (k-NN) method is an example of the neighborhood approach while singular value decomposition (SVD) and variational Bayesian principal component analysis (VBPCA) are examples of latent factor models. SVD and VBPCA are based on matrix factorization and transform both links and users to the same latent factor space. This latent factor space then tries to explain the votes on factors automatically inferred from user feedback. (Ricci et al., 2011, p. 146)

The accuracy of recommendations may significantly improve when external information is involved (e.g. timestamps), in addition to users and links. For example, time-aware SVD has been shown to improve the prediction accuracy on the Netflix dataset greatly (Ricci et al., 2011, p. 160), but the Reddit dataset does not contain any temporal data so time-aware factor models were not seriously considered. However, the Reddit dataset does include one variable that can be considered additional information: the *subreddits*. The subreddits are basically subforums containing similarly themed links, e.g. links related strictly to politics or science. In this thesis, this subreddit data was used in an attempt to improve the performance of the k-NN model (See Section 5.2.1 for details). However, the results from the experiments in Sections 5.3-5.5 indicate that the other k-NN models performed better.

2.2 Terminology and Notation

For more formal review, some definition of notation is needed. In this thesis the same notation is used as in the book *Recommender systems handbook* by Ricci et al. (2011). The set of all users will be denoted by $\mathcal{U}$, and the set of links by $\mathcal{I}$. The sizes of the sets $\mathcal{U}$ and $\mathcal{I}$ are $|\mathcal{U}| = m$ and $|\mathcal{I}| = n$, correspondingly.

The set of all votes is denoted by $\mathcal{R} = \mathcal{U} \times \mathcal{I}$, and the possible values for votes as $\mathcal{S} = \{\text{downvote}, \text{upvote}\} = \{-1, 1\}$, though most of the methods used in the thesis

give estimations where $\mathcal{S} = [-1, 1]$. It is also assumed that only one vote r_{ui} can be cast by any user $u \in \mathcal{U}$ for a particular link $i \in \mathcal{I}$. To distinguish users from links, special indexing letters i, j, l will be reserved for links and u, v for users. Thus, the subset of users who have voted a particular link $i \in \mathcal{I}$ are denoted with $\mathcal{U}_i$. Likewise, $\mathcal{I}_u$ represents the subset of links voted by user $u \in \mathcal{U}$. Also, subset of links that are voted by both users $u, v \in \mathcal{U}$ are denoted as $\mathcal{I}_{uv} = \mathcal{I}_u \cap \mathcal{I}_v$. Likewise for users, i.e., $\mathcal{U}_{ij} = \mathcal{U}_i \cap \mathcal{U}_j$ for $i, j \in \mathcal{I}$.

A known vote r_{ui} indicates the preference of link i to user u, where higher value indicates stronger preference. For the Reddit dataset the only values for the known r_{ui} are -1 and 1. The predicted ratings are denoted by $\hat{r}_{ui} \in [-1, 1]$. All of the known pairs of (u, i) for which r_{ui} is known are stored in the set $\mathcal{K} = \{(u, i)|r_{ui} \text{ is known}\}$. Later on when the datasets are split into training and test sets, these are referred with subscripts such as $\mathcal{K}_{\text{train}}$.

All vectors are represented by bolded lowercase letters, such as $\boldsymbol{u}, \boldsymbol{v}$, while matrices are represented by bolded uppercase letters, e.g. $\boldsymbol{A}, \boldsymbol{B}$.

Chapter 3

Methods

This chapter describes the collaborative filtering or machine learning methods used for the vote estimation in detail. The methods presented in this chapter were chosen because of differences in accuracy, performance and complexity. Also, baseline predictor and SVD using stochastic gradient descent (Sections 3.1 and 3.3) are considered classic methods in the field of collaborative filtering (Ricci et al., 2011). Neighborhood based methods are also often used (Section 3.2), but approach the CF problem in a very different way than latent factor models. Variational Bayesian Principal Component Analysis is also introduced in Section 3.4 and is by far the most advanced method used. Finally, multiple linear regression model (MLRM) is introduced in Section 3.5 as an example of an ensemble method, which combines the classifiers.

3.1 Naive Methods and the Baseline Predictor

Methods in this section represent common methods to be applied to new datasets. Due to very large upvote/downvote ratio, it seems relevant to test how well simple classifiers succeed. An example of a classifier that contains no model is the naive classifier that simply classifies all the votes as upvotes. The value of the RMSE metric (introduced in Chapter 4) for the naive method is called the *dummy baseline*. If a model performs worse than the dummy baseline, there is either something wrong with the implementation of the model or that model is simply ill-suited to this kind of problem due to sparsity reasons, for example. Another simple method is the random classifier that labels a given vote as an upvote with a probability of P_{upvote} and as a downvote with a probability of $P_{downvote} = 1 - P_{upvote}$. The advantage of this against the naive classifier is that some of the downvotes get correctly estimated too.

While the interest in CF models is to model the interaction between users and links, much of the observed votes are due to effects associated with either users or links, independently of their interaction (Ricci et al., 2011, p.148). For example, some user might give downvotes 99% of the time or some controversial topic might get downvotes from people that otherwise vote consistently but not in this particular case. *Baseline predictors* (also known as *biases*) will encapsulate those effects.

A baseline predictor applied to the vote bias problem can be described as follows. Let μ denote the overall average value of a vote. Since around 90% of votes are upvotes (ones), the average vote will be around 0.8. This of course does not take into account

all the missing values indicated with a zero. Also, let $b_u, b_i \in \mathbb{R}$ denote the observed deviations of user u and link i, respectively, from the average. Then the predicted rating b_{ui} will be calculated as

$$b_{ui} = \mu + b_u + b_i. \tag{3.1}$$

For example, let link j be slightly less interesting than an average link, so its parameter b_j would be, for example, -0.4. Also, let the user v be very negative about everything so that he tends to rate things well below the average at $b_v = -0.5$. When $\mu = 0.8$, the given vote will be estimated to be $b_{jv} = 0.8 - 0.4 - 0.5 = -0.1$ which rounds to -1, that is, a downvote. Of course, the parameters b_u and b_i must first be estimated for each value of i and u, respectively.

The estimation can be done by solving the least squares problem

$$\min_{b*} = \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda\Big(\sum_u b_u^2 + \sum_i b_i^2\Big), \tag{3.2}$$

where the first sum term strives to find such parameters b_u and b_i that fit the given ratings the best. The second term with λ is the regularizing term that attempts to limit overfitting by penalizing the magnitudes of the parameters. Minimizing the Equation 3.2 can be done using a simple stochastic gradient descent optimization popularized by Simon Funk (Funk, 2006). First, all of the biases are initialized as random values between [-1,1] for all users $u \in \mathcal{U}$ and links $i \in \mathcal{I}$. The algorithm loops over all known votes $r_{ui} \in \mathcal{K}_{\text{train}}$ and for each vote a prediction $\hat{r}_{ui}$ is made and the corresponding error $e_{ui} = r_{ui} - \hat{r}_{ui}$ is computed. The algorithm continues by moving along the opposite direction of the gradient, i.e., for a given training case r_{ui} the update rules are:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$

where γ controls the learning rate and λ is the regularization parameter, for example, $\gamma = 0.0006$ and $\lambda = 0.2$ for the small dataset in Section 5.3.
(Ricci et al., 2011, p. 148-152)

This simple method is easy to implement and no special care was given to optimize the parameters since this is the baseline predictor and the interest was more on the more complex methods, for which the first is Singular Value Decomposition in Section 3.3.

3.2 Neighborhood Based Methods

Nearest neighbors approach estimates the behavior of the active user based on the users that are most similar to the active user or likewise find links that are similar to the voted links. Intuitively it seems reasonable that if users u and v are most alike from the set of all users, based on some criteria, then their voting behavior should be too. For example, if user u upvotes a link, then it is unlikely for user v to downvote it

and vice versa. The problem is how to measure the similarity between users and how many nearest neighbors of user u should be taken into account when estimating the behavior of u.

In this thesis, *vector cosine-based similarity* and *weighted sum of others' ratings* were used, as in (Su and Khoshgoftaar, 2009). Thus, this is not the simplest form of k-nearest neighbors (k-NN) implementation. In vector cosine similarity, the angle between the user vectors is measured and the more similar the users are, the higher the similarity value is. More formally, when $u, v \in \mathcal{U}$ and $\boldsymbol{u}, \boldsymbol{v}$ are the corresponding vectors containing user votes for all links, then the similarity between users u and v is defined as

$$w_{uv} = \cos(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{||\boldsymbol{u}||||\boldsymbol{v}||}. \tag{3.3}$$

Similar to Eq. 3.3, the cosine similarity w_{ij} can be computed for links $i, j \in \mathcal{I}$.

Now, let $\mathcal{N}_i(u)$ denote the set of closest neighbors to u that have rated link i. The classification of $\hat{r}_{ui}$ can then be performed userwise by choosing an odd number of k neighboring users from the set $\mathcal{N}_i(u)$ and classifying $\hat{r}_{ui}$ to the class that contains more votes. Similarly, the linkwise classification can be done for links in set $\mathcal{N}_u(i)$. The classification depends on how many neighbors k are chosen in total. The parameter k can be estimated through K-fold cross validation that will be introduced in Section 5.2.

However, this kind of simple neighborhood model does not take into account that users have different kind of voting behavior. Some users might give downvotes often while some other users might give only upvotes. For this reason it is good to introduce rating normalization into the final k-NN method:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv}(r_{vi} - \bar{r}_v)}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \tag{3.4}$$

Here, the term $\bar{r}_u$ denotes the average rating by user u to the items in $\mathcal{I}_u$ and is called the mean-centering term. Mean-centering is also included in the nominator for the neighbors of u. The denominator is simply a normalizing term for the similarity weights w_{uv}. This same formula can be written for link-based recommendations as follows:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij}(r_{uj} - \bar{r}_j)}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \tag{3.5}$$

Both of these formulas will be implemented in Chapter 5. (Ricci et al., 2011)

There will also be a third neighborhood model implemented, which is based on Equation 3.4 but the similarities between users are not computed between vectors containing user votes for all links, but instead on their upvote ratio per each subreddit. Since subreddits contain similar kinds of links, it is reasonable to assume that users that have similar voting behaviors inside the same subreddits, may be similar in other ways too. The main advantage with this approach is that the subreddit feature vector can be several orders of magnitude smaller than the link vote vector of a user.

3.3 Singular Value Decomposition

Singular Value Decomposition (SVD) is a popular latent factor model in CF tasks and it has been a successful approach in the Netflix contest (Bennett and Lanning, 2007). There exist several extensions of the SVD algorithm, or rather algorithms that utilize singular values, such as SVD with k-NN and SVD with kernel ridge regression (Paterek, 2007), but these are left out of the scope of this thesis.

The principle of SVD is to decompose any rectangular matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T, \tag{3.6}$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix containing the singular values d_j of $\boldsymbol{A}$ (Siltanen and Müller, 2012). The singular values d_j are nonnegative and in decreasing order such that

$$d_1 \geq d_2 \geq \cdots \geq d_{\min(m,n)} \geq 0, \tag{3.7}$$

and of these only k values, or components, are chosen to use in this classification task.

SVD can be implemented in various ways, however, in CF field there are many successful applications utilizing SVD with stochastic gradient descent popularized by Funk (2006), such as Németh and Tikk (2007); Koren (2008). SVD implementation used in this thesis is similar to baseline predictor in Section 3.1 and does not resemble equation 3.6 at all since only the k highest singular values matter. This is done to make it use less memory and to be able to use it with sparse matrices. Matrix factorization models map users and links to a joint latent factor space of dimensionality k such that the interactions between users and links are modeled as inner products in $\mathbb{R}^k$. For example, some factors might measure some obvious dimensions such as a link relating to cats or users having some common feature, but most are probably completely uninterpretable. Of course with the particular censored Reddit dataset used in this thesis, there is no way to interpret any of the factors as being something obvious, but it is likely that some of the factors could be interpreted if there was real data of the contents of the links or messages posted by the users.

Let each link i be associated with a vector $\boldsymbol{q}_i \in \mathbb{R}^k$, and each user be associated with a vector $\boldsymbol{p}_u \in \mathbb{R}^k$. The elements in $\boldsymbol{p}_u$ measure the extent (positive or negative) to which the user u possesses the corresponding factors and similarly the elements in $\boldsymbol{q}_i$ measure the factors for link i. It is worth pointing out explicitly that these factors $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ are different for both link and users, i.e., link factors measure different aspects of the data than the user factors.

The resulting dot product $\boldsymbol{q}_i^T \boldsymbol{p}_u$ represents the overall interest of user u into the characteristics of link i. The final rating also includes the baseline predictors from Section 3.1 that depend only on the link or user such that the vote is predicted by rule.

$$\hat{r}_{ui} = \mu + b_i + b_u + \boldsymbol{q}_i^T \boldsymbol{p}_u \tag{3.8}$$

The model parameters $(b_u, b_i, \boldsymbol{p}_u, \boldsymbol{q}_i)$ can be learned by minimizing the regularized squared error

$$\min_{b*,\boldsymbol{q}*,\boldsymbol{p}*} = \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i - \boldsymbol{q}_i^T \boldsymbol{p}_u)^2 + \lambda(b_u^2 + b_i^2 + ||\boldsymbol{q}_i||^2 + ||\boldsymbol{p}_u||^2). \quad (3.9)$$

This minimization is usually done using alternating least squares, or stochastic gradient descent as in this thesis. The method will be similar to what was used for the two parameters in Section 3.1, but includes update rules for the 2 additional vectors $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$. Again, the error term is notated as $e_{ui} = r_{ui} - \hat{r}_{ui}$. Thus, the four update rules are

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$
- $\boldsymbol{q}_i \leftarrow \boldsymbol{q}_i + \gamma \cdot (e_{ui} \cdot \boldsymbol{p}_u - \lambda \cdot \boldsymbol{q}_i)$
- $\boldsymbol{p}_u \leftarrow \boldsymbol{p}_u + \gamma \cdot (e_{ui} \cdot \boldsymbol{q}_i - \lambda \cdot \boldsymbol{p}_u)$

This method could be improved by dedicating separate learning rates $(\gamma_1 - \gamma_4)$ and regularization $(\lambda_1 - \lambda_4)$ to each type of learned parameters, i.e., to user biases, link biases and factors themselves. This kind of strategy is described in more detail in Takács et al. (2008). (Ricci et al., 2011, p. 151-152)

3.4 Variational Bayesian Principal Component Analysis

Variational Bayesian Principal Component Analysis (VBPCA) is the most advanced and complex method used in this thesis. It is based on much simpler Principal Component analysis described in Section 3.4.1 and probabilistic PCA (PPCA) in Section 3.4.2. These two methods are not implemented in this thesis, rather, they serve as the introduction to the VBPCA. This section is mostly based on (Ilin and Raiko, 2008), which serves as the technical report for the Matlab toolbox for variants of probabilistic PCA algorithms in the presence of missing values that was developed by the authors of the paper at the Department of Information and Computer Science of Aalto University. Good references for using VBPCA were also the master's theses of Tommi Vatanen (Vatanen, 2012) and Polina Noeva (Noeva, 2012).

3.4.1 Principal Component Analysis

Principal Component Analysis (PCA) is a technique that can be used to compress high dimensional vectors into lower dimensional ones and has been extensively covered in literature (See e.g. Jolliffe (2002)). Assume we have N data vectors of dimension d represented by $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N$ that are modeled as

$$\boldsymbol{y}_j \approx \boldsymbol{W}\boldsymbol{x}_j + \boldsymbol{m}, \quad (3.10)$$

where $\boldsymbol{W}$ is a $d \times c$ matrix, x_j are $c \times 1$ vectors of principal components and $\boldsymbol{m}$ is a $d \times 1$ bias vector such that the relation $c \leq d \leq N$ is assumed. PCA finds $\boldsymbol{W}, \boldsymbol{x}_j$ and $\boldsymbol{m}$ such that they minimize the reconstruction error on cost function

$$C = \sum_{j=1}^{N} ||\boldsymbol{y}_j - \boldsymbol{W}\boldsymbol{x}_j - \boldsymbol{m}||^2. \tag{3.11}$$

The solution for PCA is the unique principal subspace such that the column vectors of $\boldsymbol{W}$ are mutually orthonormal and, furthermore, for each $k = 1, \ldots, c$, the first k vectors form the k-dimensional principal subspace. The principal components can be determined in many ways, including singular value decomposition, least-square technique, gradient descent algorithm and alternating W-X algorithm. All of these can also be modified to work with missing values. More details are discussed in (Ilin and Raiko, 2008).

It is worth noting here that SVD and PCA are very closely related. Let $\boldsymbol{X}$ be a data matrix that has been preprocessed to have a zero mean. Principal components of $\boldsymbol{X}$ can be computed through the eigenvectors of its covariance matrix $\boldsymbol{X}\boldsymbol{X}^T$. Since the covariance matrix is symmetric, the matrix is diagonalizable and the eigenvectors can be normalized to be orthonormal:

$$\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T. \tag{3.12}$$

Assume $\boldsymbol{X}$ has the SVD expansion $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$ according to Eq. 3.6. Then

$$\begin{aligned} \boldsymbol{X}\boldsymbol{X}^T &= (\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T)(\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T)^T \\ &= (\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T)(\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^T) \qquad (\text{since } \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}) \\ &= \boldsymbol{U}\boldsymbol{D}^2\boldsymbol{U}^T \\ &= \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T. \end{aligned} \tag{3.13}$$

This means that the square roots of the eigenvalues of $\boldsymbol{X}\boldsymbol{X}^T$ are the singular values of $\boldsymbol{X}$, and the columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$.

3.4.2 Probabilistic PCA

For PCA, there is a notable absence of an associated probabilistic model for the observed data, which probabilistic PCA (PPCA) includes (Tipping and Bishop, 1999). Other advantages of PPCA include well-founded regularization, model comparison, interpretation of results and extendability (Ilin and Raiko, 2008). The PPCA model includes a Gaussian noise term explicitly in the PCA model (3.10) as

$$\boldsymbol{y}_j = \boldsymbol{W}\boldsymbol{x}_j + \boldsymbol{m} + \boldsymbol{\epsilon}_j \tag{3.14}$$

where principal components $\boldsymbol{x}_j$ and the noise $\boldsymbol{\epsilon}_j$ have a Gaussian prior distribution as

$$p(\boldsymbol{x}_j) = \mathcal{N}(\boldsymbol{x}_j; 0, \boldsymbol{I}), \quad p(\boldsymbol{\epsilon}_j) = \mathcal{N}(\boldsymbol{\epsilon}_j; 0, \sigma^2\boldsymbol{I}). \tag{3.15}$$

Here, $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes normal probability density function (pdf) over variable $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$, covariance $\boldsymbol{\Sigma}$ and $\boldsymbol{I}$ is the identity matrix. The variables $\boldsymbol{x}_j$ and noise

terms $\boldsymbol{\epsilon}_j$ are assumed to be independent, identically distributed (i.i.d) and mutually independent. The parameters of the model include $\boldsymbol{W}, \boldsymbol{m}$ and v and is expressed as

$$p(\boldsymbol{y}_j \mid \boldsymbol{x}_j, \boldsymbol{W}, \boldsymbol{m}, v) = \mathcal{N}(\boldsymbol{y}_j; \boldsymbol{W}\boldsymbol{x}_j + \boldsymbol{m}, \sigma^2 \boldsymbol{I}) \tag{3.16}$$

The model can be identified by finding the maximum likelihood (ML) estimate with expectation maximization (EM) algorithm. For more details on the algorithm, see Ilin and Raiko (2008). While PPCA is less prone to overfitting than the simplest least-squares algorithms, it can overfit, especially with unreasonably large number of principal components.

In chapter 5, no experiments were run using this algorithm, since VBPCA algorithm always seemed to outperform simpler PPCA in the random experiments made during the early parts of this thesis process.

3.4.3 Variational Bayesian PCA (VBPCA)

PPCA seems to suffer from overfitting in some cases and a common way to cope with it is to penalize parameter values that lead to increased complexity in explaining the data. In the Bayesian formulation this means introducing a prior over the model parameters in addition to PPCA model's (3.14)-(3.15) so that

$$p(\boldsymbol{m}) = \prod_i \mathcal{N}(m_i, \mu, w_m), \quad p(\boldsymbol{W}) = \prod_{k=1}^{c} \mathcal{N}(\boldsymbol{W}_k; 0, w_k \boldsymbol{I}). \tag{3.17}$$

The model (3.17) uses the same priors for each column $\boldsymbol{W}_k$ in the matrix $\boldsymbol{W}$ and includes hyperparameters w_m and w_k that can also be updated during learning. Hyperparameter μ for the mean is also used for $\boldsymbol{m}$. Parameter w_k is used for determining the right number of principal components in the model, resulting in w_k values tending to zero when the evidence for the corresponding k-th principal component for reliable data modeling is weak. This kind of technique is referred to as *automatic relevance determination* (ARD) in the machine learning literature (Bishop, 2006).

Variational Bayesian Principal Component Analysis (VBPCA) is based on variational Bayesian inference first introduced in Bishop (1999), as one of the central issues with using PCA is that of choosing the appropriate number of components. Let us consider the ML estimation of the hyperparameters $\boldsymbol{\xi} = (v, w_k, w_m)$ in the model defined in (3.14) - (3.17). The variational view of the EM algorithm requires the computation of the posterior $p(\boldsymbol{\theta}|\boldsymbol{Y}, \boldsymbol{\xi})$ on the E step.

The E-step is modified to update the approximation $q(\boldsymbol{\theta})$ in order to minimize the cost function

$$\begin{aligned} C(q(\boldsymbol{\theta}), \boldsymbol{\xi}) &= \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \boldsymbol{Y}|\boldsymbol{\xi})} \, d\boldsymbol{\theta} \\ &= \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\boldsymbol{Y}, \boldsymbol{\xi})} \, d\boldsymbol{\theta} - \log p(\boldsymbol{Y}|\boldsymbol{\xi}) \end{aligned} \tag{3.18}$$

On the M-step the likelihood $p(\boldsymbol{Y}|\boldsymbol{\xi})$ is maximized using the posterior approximation $q(\boldsymbol{\theta})$ (Noeva, 2012). Computationally convenient form for PCA model is

$$q(\boldsymbol{\theta}) = \prod_{i=1}^{N} q(m_i) \prod_{i=1}^{N} q(\boldsymbol{w}_i) \prod_{j=1}^{D} q(\boldsymbol{x}_j) \tag{3.19}$$

Now, the object is to find the distribution $q(\boldsymbol{\theta})$ of form (3.19) such that the cost function $C(q(\boldsymbol{\theta}), \boldsymbol{\xi})$ is minimized.

The EM algorithm for VBPCA includes the following steps (Noeva, 2012):

1. Initialize the hyperparameters $\boldsymbol{\xi}$
2. E step: update alternatively one factor of $q(\boldsymbol{\theta})$ while keeping the other factors fixed to minimize the cost function (3.18)
3. M step: compute new values for hyperparameters $\boldsymbol{\xi}$ to increase the likelihood $p(\boldsymbol{Y}|\boldsymbol{\xi})$ with the distribution $q(\boldsymbol{\theta})$ used as it was the true posterior density function $p(\boldsymbol{\theta}|\boldsymbol{Y}, \boldsymbol{\xi})$.

For a curious reader, the steps and the exact update rules for all parameters are explained in more detail in Appedix E (VBPCA With Fully Factorial Approximation) of (Ilin and Raiko, 2008).

3.5 Multiple Linear Regression Model

Models can be composed of multiple classifiers that complement each other so that the resulting model gives better accuracy than any one classifier by itself. There are many techniques for combining multiple classifiers, such as *voting*, *bagging* and *stacking* (Alpaydin, 2004). Voting corresponds to taking a linear combination of the classifiers, also known as ensembles, by giving a weight to each one. Better classifiers get higher weights and contribute more to the final classification. Bagging is a voting method that trains the classifiers on slightly different training sets thus making them different. Stacking is a technique that combines base-learners through a combiner system, which is a learner itself, so that its output does not have to be linear.

In this thesis, the classical *multiple linear regression model* (MLRM) is used due to its simplicity. MLRM is the classical regression model for more than one explanatory variable that is explained in (Moore and McCabe, 2006), for example. Let $x_1, \ldots, x_p$ be the explanatory variables and y the response variable. The statistical model for multiple linear regression is then

$$y = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \cdots + \beta_p x_p + \epsilon, \tag{3.20}$$

where $\beta_0, \ldots, \beta_p$ are the regression coefficients and ϵ is the noise term. Another way to look at this model is to consider the response variable y to be the final classification that linearly depends on p classifiers $x_1, \ldots, x_p$ with weights $\beta_0, \ldots, \beta_p$, where β_0 is the weight for the naive classifier. When y is represented as a vector containing all the estimates, the model can be represented as

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_1 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i, \tag{3.21}$$

for $i = 1, 2, \ldots, |\mathcal{K}_{\text{test}}|$ (See 2.2 for notation). Additionally, the deviations ϵ_i are i.i.d and pair-wise independent from the normal distribution $\mathcal{N}(0, \sigma^2)$. The regression

coefficients $\beta_0, \ldots, \beta_p$ can be solved using the method of least squares, i.e., minimizing the quantity

$$\sum_{i=1}^{|\mathcal{K}_{\text{test}}|} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_1 x_{i2} - \cdots - \beta_p x_{ip}). \tag{3.22}$$

Since regression analysis is such an integral part of any statistics toolbox for different programming languages or statistics textbook, this presentation will not attempt to explain the MLRM principle any further.

Once the regression coefficients have been solved, the final predictions can be computed by the linear combination

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_1 x_{i2} + \cdots + \beta_p x_{ip}. \tag{3.23}$$

The values of regression coefficients $\beta_0, \ldots, \beta_p$ also reveal how much each classifier contributes to the final prediction.

Chapter 4

Model Evaluation

This chapter introduces the evaluation metrics used in this thesis and some general model evaluation methods. Evaluating the results is very important in this kind of application to prevent models from overfitting to the training set, thus decreasing the performance on the test set. Also, searching for the best parameters for the methods, particularly for SVD and k-NN, using K-fold cross validation is essential.

4.1 Evaluation Metrics

Consider the problem of predicting the votes r_{ui} for the Reddit dataset. This section describes how methods can be compared using different metrics. The simplest metric is called *accuracy* and it determines how many of the total estimates are predicted correctly, i.e., number of pairs (u, i) such that $\hat{r}_{ui} = r_{ui}$. This is a rather noninformative metric for the Reddit dataset, since the number of upvotes is 7-8 times higher than the amount of downvotes, so simply predicting all votes being upvotes gives accuracy of 0.86-0.90 depending on the dataset (see Table 5.4 for the properties of different datasets).

The Root Mean Squared Error (RMSE) is probably the most popular metric used in evaluating the accuracy of predicted ratings (Ricci et al., 2011, p. 273). It was also chosen as the measure in the Netflix prize competition (Netflix, 2009), where the prize was offered to the first team being able to decrease RMSE of Netflix's own algorithm by 10%. The RMSE between the predicted ratings $\hat{r}_{ui}$ and actual ratings r_{ui} for the known user-link pairs (u, i) in test set $\mathcal{K}_{\text{test}}$ is the defined as

$$RMSE = \sqrt{\frac{1}{|\mathcal{K}_{\text{test}}|} \sum_{(u,i)\in\mathbb{K}} (\hat{r}_{ui} - r_{ui})^2}. \tag{4.1}$$

By squaring the difference of predicted and actual rating, more emphasis is put to larger errors. In Netflix ratings, this works well since the ratings by users are on scale ranging from 1 to 5 and the recommendation algorithm by the Netflix gives predicted movie ratings for users as full and partial stars, i.e., not integers but real numbers with one decimal (3.7 for example). However, in binary classification case such as this thesis' upvote/downvote classification, there will be only 2 classes, nothing inbetween the classes. Still, most of the algorithms introduced in Chapter 3 model the error as

a real number during stochastic gradient descent, and VBPCA in particular displays the RMSE error during every iteration. This means that while this *softer* RMSE is a good measure for the convergence of the algorithms, the real classification of upvotes and downvotes need to be *hard*, i.e., $\hat{r}_{ui} \in \{-1, 1\}$. For this reason, every algorithm was measured with *soft RMSE*, meaning imputing the estimates into Equation 4.1, and *hard RMSE*, which rounds the estimates to the nearest class and only then applies RMSE.

Soft RMSE was used in (Poon et al., 2011) and gave the only reference measurement that could be found for this dataset. In their coursework, they combined several algorithms, including k-Nearest-Neighbors, SVD, K-means and "Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo", and the linear combination of the models. The best score they got was RMSE of 0.491433, which became a goal in this thesis. In the end, it was clear that VBPCA was able to beat this result by itself, but still did not dominate all the other metrics for methods used in this thesis. Also, they do not justify the usage of RMSE as their only metric well enough. In general, there are four possible outcomes between the real votes and predicted votes, as shown in Table 4.1.

		Predicted vote:	
		Upvote	Downvote
Actual vote:	Upvote	True-Positive (tp)	False-Negative (fn)
	Downvote	False-Positive (fp)	True-Negative (tn)

Table 4.1: Confusion matrix, listing the four possible outcomes of Reddit dataset classification.

Some commonly used quantities can be computed from the table values, namely:

$$\begin{aligned} \text{Precision} &= \frac{\#\text{tp}}{\#\text{tp} + \#\text{fp}} \\ \text{Recall (True Positive Rate)} &= \frac{\#\text{tp}}{\#\text{tp} + \#\text{fn}} \\ \text{False Positive Rate} &= \frac{\#\text{fp}}{\#\text{fp} + \#\text{tn}} \end{aligned} \tag{4.2}$$

Two of the most common ways to compare these quantities are called precision-recall curves and Receiver Operating Characteristic (ROC), both of which measure the proportion of preferred links that are actually recommended. While precision-recall curves compare precision with recall that emphasizes the proportion of recommended items that are preferred, ROC curves emphasize the proportion of items that are not preferred but end up being recommended anyway. (Ricci et al., 2011, p. 275). To select whether to use precision-recall or ROC is based on the domain. In this Reddit recommendation task it is more important to minimize false positive rate, to prevent users seeing links that they may find offensive or boring.

4.2 K-Fold Cross Validation

The problem with unregularized machine learning models consisting of some parameters $\boldsymbol{\theta}$ is to choose the best performing parameters value $\boldsymbol{\theta}_{\text{best}}$. Usually a more complex model gives better results for the training set, for example if the scalar parameter $\theta = k$ is the number of principal components, the higher values of k gives better results simply because the model is overfitting the training set. However, this does not mean that the model is necessarily the best choice for the test set, since the data structure is slightly different from the training set. This problem is similar to the often cited problem of finding the best value k for fitting a polynomial of degree k to a set of points.

In K-fold cross validation, the dataset X is randomly divided into K equal-sized parts $X_i, i = 1, \ldots, K$ and the model is trained K times such that each time a certain dataset X_i is chosen as a validation set and the rest are combined into the actual training set. Then, the validation set errors are used in determining which model, or model parameters, would perform best on the test set. It is also imporant to make sure that class prior probabilities are similar between the training set and the validation set. This is called *stratification*. (Alpaydin, 2004, p. 331)

With the Reddit dataset, stratification is practically ensured because the dataset is so large that this happens naturally when dividing the votes randomly into the training and validation sets (see Section 5.2).

Chapter 5

Experiments

The first section in this chapter introduces the full Reddit dataset, its properties and the preprocessing that was performed. Section 5.1.2 examines the three different subsets of the full Reddit dataset and how the parameters were chosen. Section 5.2 examines the training sets through K-fold cross validation and attempts to find the best number of neighbors k for all the different k-NN tests and also the number of components for SVD that should give the best results for the test set. The next three Sections (5.3-5.5) explain how the experiments were run and what were the results. Finally, Section 5.6 examines how the estimation accuracy might have depended on other factors, such as the ratio of upvotes per downvotes of a user.

5.1 Full Reddit Dataset

Reddit dataset originated from a topic in social news website Reddit (King, 2010). It was posted by a software developer working for Reddit in early 2011 in hopes of improving their own recommendation system. The original dataset consists of 23,091,688 votes from 43,976 users over 3,436,063 links in 11,675 subreddits (see the definition of a subreddit from Section 2.1). A snapshot of preprocessed dataset can be seen in Table 5.1 along with the possible values for the different variables in Table 5.2.

The users in the dataset represent almost 17% of the total votes on the site in 2008, which means that the users represented in the dataset are very active in the whole Reddit community. All of these users also gave their permission to use their votes in

User id	Link id	Subreddit id	Vote
52133	1642985	2	-1
11720	1613099	3	1
11720	2139739	3	1
12070	1577844	3	1
⋮	⋮	⋮	⋮

Table 5.1: A snapshot of some rows of the Reddit dataset after changing the labels from strings into numbers and sorting subreddit-wise. There are over 23 million rows like this.

Users:	$\{1, \ldots, 43976\}$
Links:	$\{1, \ldots, 3436063\}$
Subreddits:	$\{1, \ldots, 11675\}$
Votes:	$\{-1, 1\}$

Table 5.2: All the different values possible in each column.

research (King, 2010). This means that this subset of users might not describe the whole userbase of Reddit accurately, but this cannot be proven either way from the dataset and in any case these experiments could be thought to be accurate to *at least* this subset of users.

The original dataset has over 23 million rows where the first three columns contain 32 hexadecimal digits, unique to specific link, user, or subreddit. The fourth column indicates whether the user gave an upvote -1 or an upvote 1. No other values are possible as shown in Table 5.2, which also means there is no zero value to indicate that the user has indeed seen the link, but has not given a vote either way. This means that there is no way to know for sure whether any of the links are actually new to the user, which leads to the assumption on *missing-data mechanism*.

There are four types of missingness mechanism formalized in Rubin (1987), of which *missingness at random* (MAR) is assumed for the Reddit dataset. MAR means that *given the observed data, the missingness mechanism does not depend on the unobserved data* and thus, there is no general model used for the missing data in this thesis. One can ask how much does it really matter that it is not known if a user has deliberately decided not to vote a link that he has seen? Due to the way the website Reddit works, there are several factors that would make it highly likely that a huge majority of unvoted links are such the user has not seen. First, the amount of links in the full Reddit dataset is so large that no average user would have the time and effort to skim through all of those. This is supported by the fact that the most productive user in the preprocessed datasets has voten 78478 links (on average 215 links per day for one year) while the median was only 63 for the same dataset (see Table 5.6). Second, links stay on the surface only for a couple of hours or days at the maximum, so if a user does not visit the website for a couple of days, many links will remain unseen. This is especially true for the average new users that probably will not be digging through the much older links, unless maybe the most popular ones. In conclusion, for some active users skimming through huge amounts of links without giving many votes, this estimation will probably not give very good results and would greatly benefit from having the additional information on links the user chose not to vote. Then again, it is rather justified to assume that the estimation does work for the users *on average*.

A Very important aspect of the dataset is that while the links in Reddit website always contain some topic and text, this data contains none. It is not possible to see what kind of keywords the links contain, which renders content-based recommendation systems completely useless. However, different kinds of CF techniques are applicable to this data as mentioned in Section 2.1.

The original idea behind releasing the dataset was to improve their community recommendation system, i.e., to help users to find similar user groups or communi-

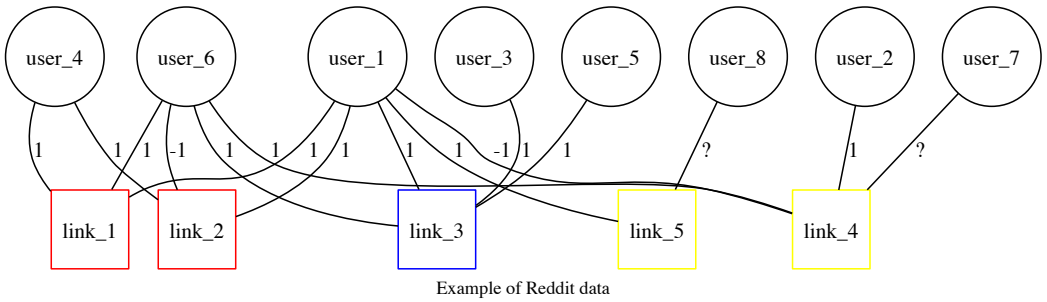

Figure 5.1: Bipartite graph representation of Reddit data. Subreddits are visualized as colors on the links (square vertices). The vote values are represented as numbers on the arcs, where "?" means no vote has been given.

ties, not simply to recommend them some new subreddit that they might think is interesting. In short, the purpose was to cluster the users. During the early preparation of this thesis, several approaches were considered to this dataset. The interesting questions, in no particular order, included:

- **Given a link, will the user upvote or downvote it?**
- What is the most relevant (new) link for the user?
- What is the most relevant (new) subreddit for the user?
- Who are the most similar users to the given user?
- Can the users be grouped in a meaningful way based on similarities and subreddits?

Eventually, the first question became the topic of this thesis and a solution for the fourth question was provided when experimenting on k-NN method, though the cosine distance (Eq. 3.3) was the only similarity measure used in this thesis.

5.1.1 Preprocessing

The original identifiers of the variables were 32 characters long strings due to data obfuscation reasons. Clearly, identifiers were too long so the first step in the preprocessing phase was to replace the original ids for the 3 different variables, users, links and subreddits, into integer numbers to help processing the data and also to save memory. This phase was implemented in the Python programming language, because of its very useful dictionary data structure. After this phase, the integers in the different columns have the values as in table (5.2). Example rows of the data are presented in table (5.1).

The dataset can be visualized as a bipartite undirected graph (figure 5.1), where the circle vertices have an edge only to link vertices, represented as squares, and

Dataset	User cutoff	Link cutoff	Subreddit cutoff
Small	1	500	1
Mid	135	135	135
Big	4	4	4

Table 5.3: cutoff values

vice versa. Subreddits can be represented as the color of square vertex and the edges between vertices are the vote values, either 1 or -1. The research question is visualized in the graph as the edges with a question mark *?* to be estimated from the known edges of the data.

Even though no graph-theoretic approaches are used in the classification problem in this thesis, this visualization is particularly useful in explaining the preprocessing and the concept of *core of the data*. In graph theory, the degree of a vertex v equals to the number of edges incident to v. If the degree of any vertex is very low, there may not be enough data about the vertex to infer the value of an edge, meaning the vote, where this vertex is in the either end. An example of this can be found in figure (5.1) between *link_4* and *user_7*, where the degree of *user_7* is 1 and the degree *link_4* is 3. Clearly this is a manifestation of the *new user problem* (Adomavicius and Tuzhilin, 2005), meaning that the user would have to vote more links and the link would have to get more votes in order to accurately estimate the edge. These kinds of new users and unpopular links make the estimation task very difficult and thus should be pruned out of the data. This can be done by using *cutoff* values such that all the corresponding vertices having a degree below the cutoff value are pruned out of the data. With higher cutoff values, only a subset of the data called *core of the data* remains, which is similar to the idea of k-cores introduced by Seidman (1983). This subset contains the most active users who have voted a large part of all the remaining links, and the most popular links which are voted by almost every user. Subreddits are also pruned out using proper cutoff values.

5.1.2 Preparing Different Sized Datasets

In order to test the algorithms with datasets that have a different sparsity level, the preprocessed dataset was pruned with three different sets of cutoff values that can be seen in Table 5.3.

The idea was to create the biggest dataset with very low cutoff values such that per each user or link there are at least four observations so links can be split between training set and test set. The smallest dataset in turn was pruned with higher link cutoff values, resulting in a smaller dataset that can fit into the computer's memory, even when expressed as a full matrix. This is exactly the same dataset that was used in Poon et al. (2011). For the middle dataset, or "mid", the cutoff values were set so that its size would lie in the middle between the sizes of the big and small datasets on log10-scale and also its cutoff values were chosen to be equal to each other.

The pruning algorithm worked by removing all subreddit vertices that had less edges than the cutoff values, then doing the same for user nodes and link nodes. The algorithm only stopped when it did not remove any type of a node during the whole

Dataset	Users	Links	Subreddits	Votes	Sparsity	Ratio of upvotes
Small	24,528	3,246	28	2,092,043	0.0263	0.9017
Mid	7,973	22,907	86	5,990,745	0.0328	0.9015
Big	26,315	853,009	2,556	17,349,026	0.000773	0.8688
Full	43,976	3,436,063	11,675	23,079,454	0.000153	0.8426

Table 5.4: Comparison of the training sets.

iteration. This ensured that the cutoff values really were valid on the pruned dataset, instead of naively removing all the different kinds of nodes with corresponding cutoff values during one iteration. This would have resulted in some nodes having less edges than the cutoff values proposed. No matter what cutoff values were chosen, the algorithm seemed to take at most 10 iterations, or about 20-60 seconds of computer time.

The resulting dataset was then immediately split into a *training set* and a *test set*, such that the training set contained about 90% of the votes and the test set around 10%, respectively. All the models were trained and model parameters estimated using only the data in the training set and the test set was only used for getting the final results. The splitting algorithm worked userwise, i.e., it randomly divided a user's votes between the training set and test set for all users such that at least one vote was put into the test set, regardless of the split ratio. This means that it was guaranteed that for each user there was at least one vote in the test set for classification, even if the user had given only a few votes. For example, in the largest dataset with user cutoff at 4, one was randomly put into test set and the rest three into training set. If a user had given 101 votes, 11 were in the test and 90 in the training set. Otherwise there would have been a large amount of less productive users, for whom all of their votes were either in the training set (not possible to test the estimation) or in the test set (just guessing the classification, i.e., classifying all votes as upvotes like a naive predictor).

The resulting training set properties are described in Table 5.4, where there are several interesting points to be seen. First, the number of subreddits gets diminished when the datasets get smaller. This means that the most relevant links for the small dataset are spread into only two dozen subreddits, which seems reasonable since there are around the same amount of default subreddits for all the users. Second, the big dataset seems to be around 34 times more sparse than the smallest, while the middle one is the least sparse. This results from the fact that the cutoff values chosen by Poon et al. (2011) were very link-heavy, while the mid dataset was pruned using equal values for the three cutoff values. For this same reason, mid dataset has less users than links, similar to the full and big dataset, but contrary to the small dataset. So in short, it has properties from both small and big dataset.

The resulting test set properties are described in Table 5.5, where it is apparent that the ratios of upvotes stay very close to the training set values. In general the estimation of downvotes is a lot more difficult than upvotes. This is partly due to the fact that downvotes are rarer and thus the prior probability for upvotes is around six to nine times higher than for the downvotes so the prior is always on the side of

Dataset	Votes	Ratio of upvotes
Small	221828	0.9032
Mid	661903	0.9018
Big	1915124	0.8687

Table 5.5: Comparison of the test sets.

		Mean	Median	Std	Max
Small dataset:	Users	68.6	11	180.1	2798
	Links	518.0	480.5	189.4	2730
Mid dataset:	Users	603.6	278	1026.7	15462
	Links	210.0	165	136.9	1908
Big dataset:	Users	486.6	63	1929.8	78478
	Links	150.1	5	45.5	2707

Table 5.6: Properties of votes given by users and votes for links of all datasets.

upvotes. This also means that the dataset is much more sparse when using only the downvotes.

The histograms of users and links in Figure 5.2 are also rather interesting. It is apparent that all these distributions have a thin tail, more so for the big dataset, and some of the figures seem to be truncated from the left, which is due to preprocessing. For example, the link histogram for the small dataset starts after around 500 links whereas the user histogram seems to start at 1. This can be simply explained by the cutoff values described in Table 5.3. Some properties of histograms in Figure 5.2 are shown in Table 5.6.

5.2 Stratified 5-fold Cross Validation

Two of the methods used in this thesis have a hyperparameter that needs to be explicitly defined in order to run the tests. For choosing the best possible value for this hyperparameter, or at least some very close approximation that is well justified, 5-fold cross validation is used. The number of folds K in K-fold cross validation is typically 10 or 30 (Alpaydin, 2004, p. 331), but the Reddit dataset is so large and uniform so stratification happens naturally, justifying the use of smaller number of folds. Also, to save computational time, the number of folds was chosen to be 5.

5.2.1 Number of Neighbors for k-NN

With the Reddit dataset, the k-NN model can be implemented in various ways, depending on whether the neighbors mean the neighbors of links, users or some other feature entirely. In this thesis, the k-NN model is implemented in three ways: user-wise, linkwise and subreddit-wise (sr-wise). As mentioned in Section 3.2, the subreddit vector is significantly smaller than the link vector. This is evident from Table 5.4, where the number of elements in the link vector is 853,009 for the big dataset, but only

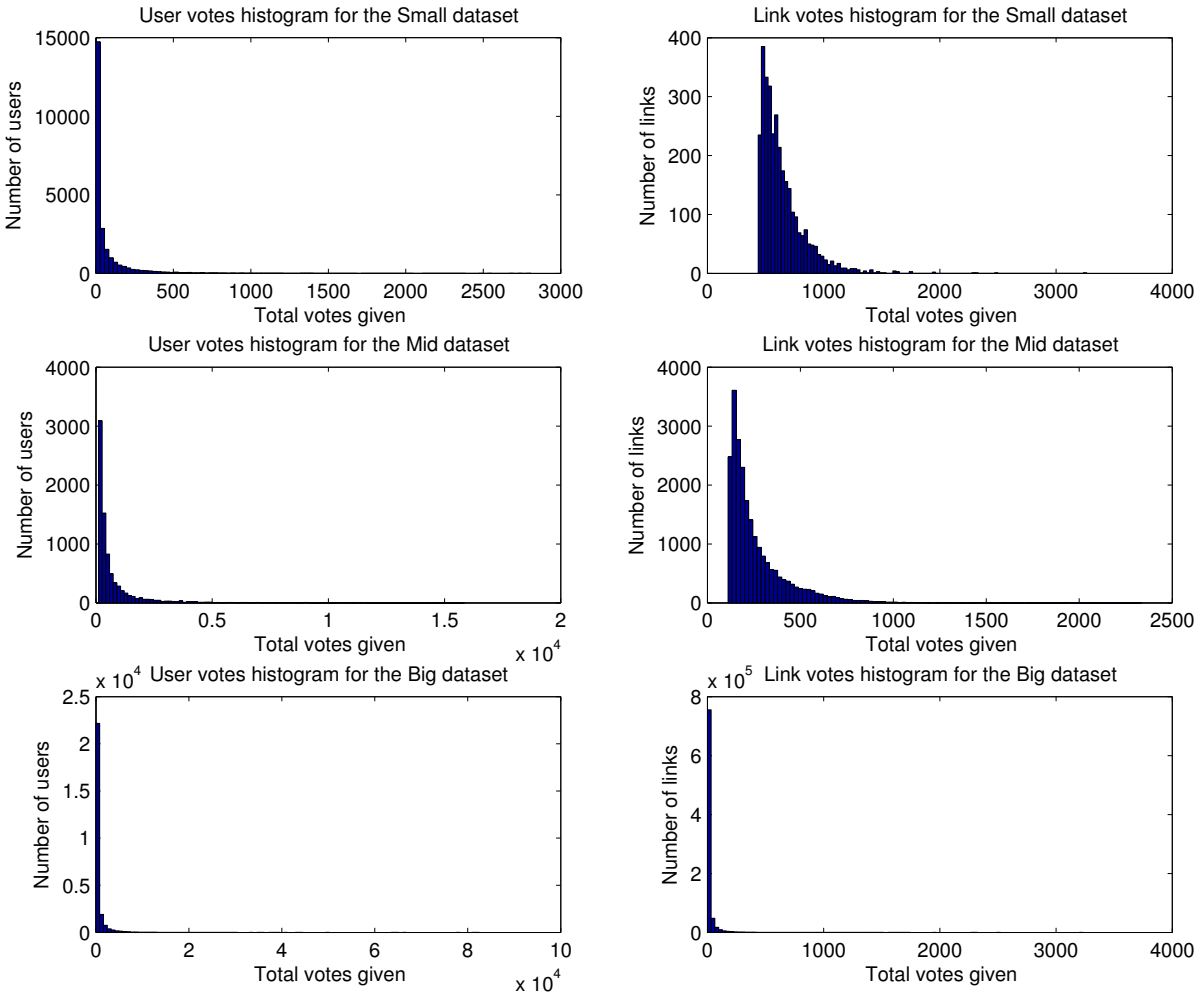

Figure 5.2: Histograms of users and links for the different datasets. The rows of pictures correspond to the small, mid and big dataset, in that order, and the columns are for users and links, correspondingly. The histograms for the big dataset are barely visible, due to their extremely thin tail.

2,556 for the subreddit vector. The computations linkwise required some significant optimizations, since computing the distance matrix between all the links would have required hundreds of gigabytes of memory and computing the necessary distances for each observed vote one at a time meant jumpin around in memory, requiring more than a week of computation time. In the end, the computation time was reduced to mere 2 hours on a regular laptop requiring less than 8 GB of memory.

The performance of the k-NN model with differing number of neighbors was evaluated using 5-fold cross validation for the training set for each odd number of neighbors k between 1 and 51. Due to random splitting of the data during cross validation, there were some votes given by users to links that no other user had voted, i.e., the users did not have any neighbors. In this situation, naive model was used and the vote was approximated to be an upvote, which would be the right decision around 86-90% of the time (the corresponding dummy baseline of the dataset). In practise, there were very few votes with no neighbors for the voting users, in practise less than about 0.1%, so its effect was considered to be negligible.

The performances between different folds had very little variance, so the overall performance was measured by taking a mean of the hard RMSE metric for the validation set of all folds.

The performance for user-based and link-based recommendation can be seen in Figure 5.3 for all datasets. It seems that the performance of user-based recommendation increases as the number of neighbors increases and if k was increased beyond 51, the RMSE might get even lower. However, the increased performance is considered so little that the values given by these experiments will be used for the test set.

The results of the subreddit-based recommendation are in Figure 5.4. The sudden increase on the right-most points on each graph is due to it representing the performance using all of the available neighbors. For the small and mid datasets, it is evident that choosing three or more neighbors results in better estimation than the dummy baseline. For the big dataset even one neighbor is enough. The best choice for the number of neighbors k was chosen to be the one that produced the smallest RMSE on the validation set. The K values for small, mid and big dataset were 21,21 and 19, correspondingly.

5.2.2 Number of Components for SVD

For estimating the optimal number of components for SVD, 5-fold cross validation was also used. The first experiments were run on SVD with the number of components of: 2, 7, 15, 30. From these results it was decided to run more experiments near the value which got the lowest RMSE on the first time so the cross validation was not for all k values between 1 and 30, but for values near the potential minimum. For each experiment, the algorithm was given at most 5000 iterations but also the possibility of stopping earlier using *rmsstop*-criterion with 100 step length. This means that the algorithm will stop when either the absolute difference $|\mathrm{RMSE}_{t-100} - \mathrm{RMSE}_t| < 0.0001$ or the relative difference $|\mathrm{RMSE}_{t-100} - \mathrm{RMSE}_t|/\mathrm{RMSE}_t < 0.001$, where RMSE_t is the corresponding validation set RMSE on iteration t. It is important to note that this rmsstop with the same parameters was used for all datasets when running the baseline algorithm and SVD on the test set. Also, the learning rates and regularization

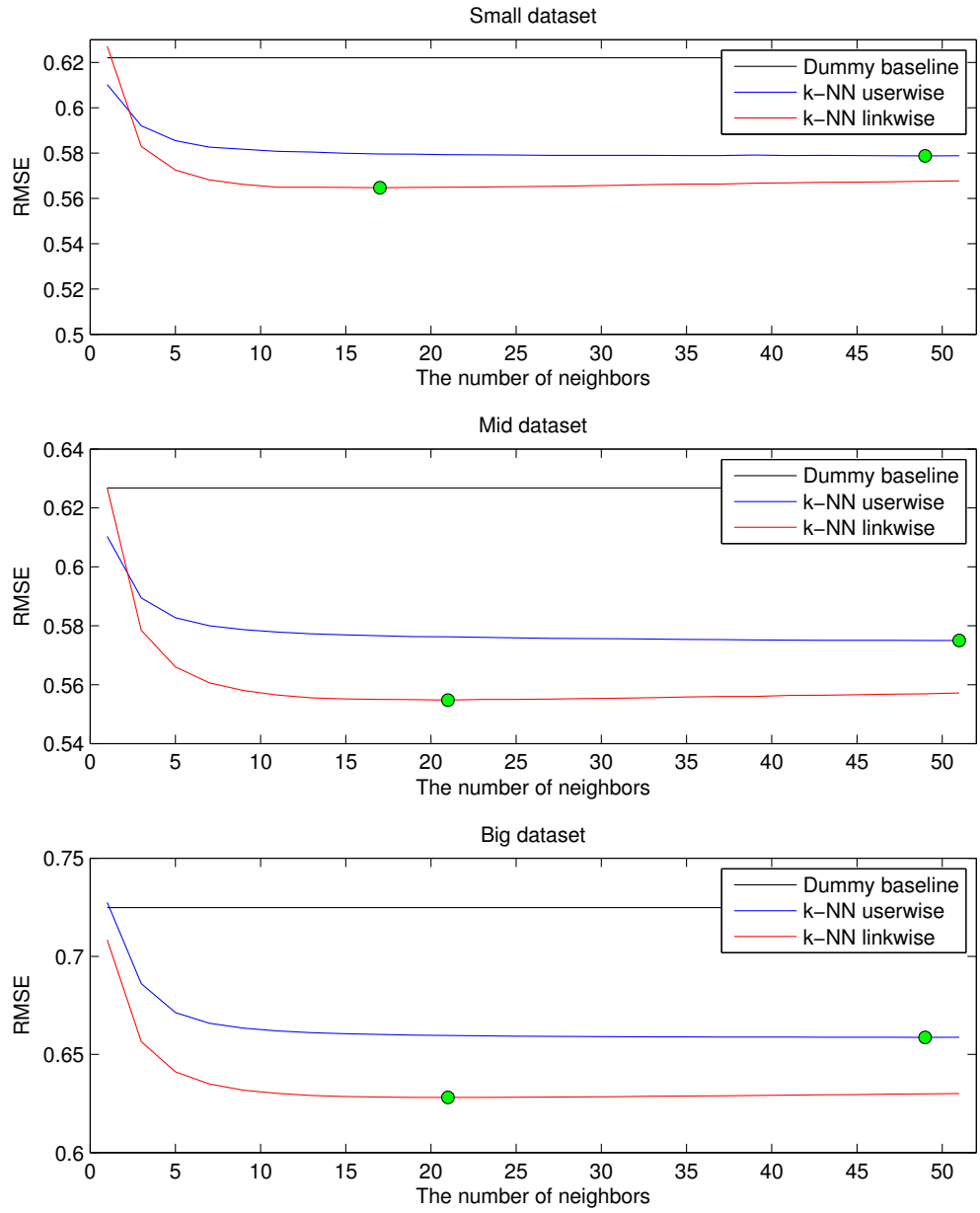

Figure 5.3: Userwise and linkwise k-NN performance on all validation sets using different number of nearest neighbors for 5-fold cross validation. The lowest RMSE (marked as a green dot) is marked on the plots. The number of neighbors to use in userwise and linkwise k-NN are 49 and 17 for small, 51 and 21 for mid, and 49 and 21 for big dataset.

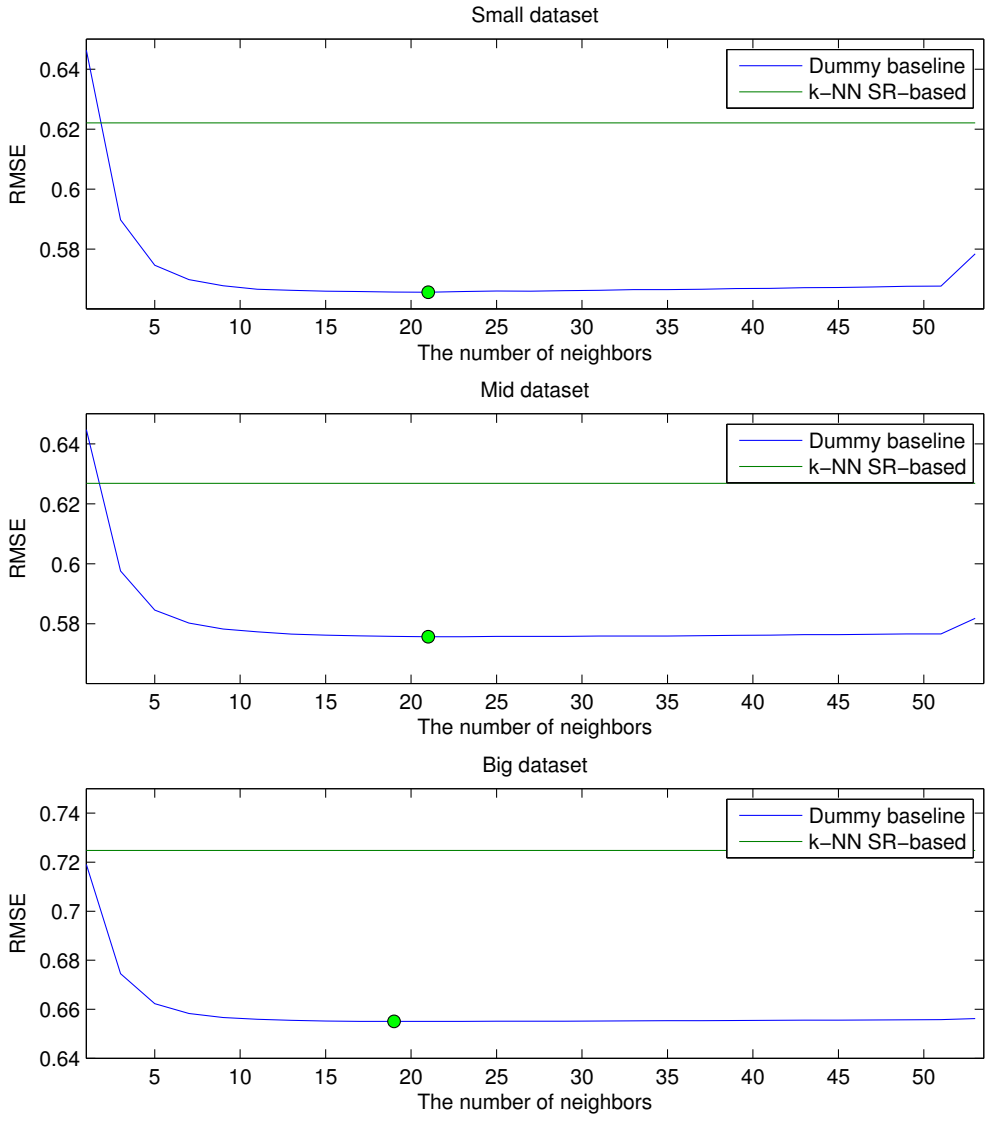

Figure 5.4: Subreddit-wise k-NN performance on all validation sets using different number of nearest neighbors for 5-fold cross validation. The lowest RMSE (marked as a green dot) is marked on the plot. These green dots are at 21, 21 and 19 for small, mid, and big datasets.

parameters in Table 5.7 are same for these crossvalidation tests and the final experiments.

Dataset	Learning rate γ	Regularization parameter λ
Small dataset	5.0×10^{-4}	0.2
Mid dataset	1.0×10^{-4}	0.2
Big dataset	2.0×10^{-5}	0.2

Table 5.7: SVD learning rates and regularization parameters used in cross validation tests and final experiments.

The results of the cross validation tests are shown in Figure 5.5 where it can be seen that the optimal number of components for the small, mid and big dataset are 4,6 and 2, correspondingly. At first it seemed that there was something wrong with experiments, e.g., with the parameters of the stochastic gradient descent, because the results meant that using less components is better for the big dataset, even though there is more potential for complexity in the dataset to be explained by higher number of components. It may be that the reason this happens is that the big dataset is simply so sparse that using more components does not really explain the data itself, instead the additional components may overfit to the structures caused by the noise inherent in the data. This hypothesis is supported by the fact that the highest number of components indicated by the cross validation results is for the mid dataset, which is the least sparse dataset.

5.3 Results for Small Reddit Dataset

This section describes the results of all the methods described in Chapter 3 on the small Reddit dataset. The models were trained on the full training set and all results are for the corresponding test set. The results of all the methods are seen in Table 5.8. For the columns *Accuracy, Class average accuracy, Downvotes*, and *Upvotes*, the higher is better. On the contrary, for columns *Soft* and *Hard RMSE*, the lower is better. Column *Upvotes* means the ratio of correctly estimated upvotes and similarly for *Downvotes*. *Class average accuracy* is the mean between these two columns. Naive and random methods were trivial to implement, since there is no real model behind, and the experiments were fast to run. As expected, random model gets some of the downvotes right by pure chance while naive model does not classify votes as downvotes at all and gives an RMSE of 0.6221, the dummy baseline for the small dataset. This is the upper limit that each real model should be able to beat.

Baseline predictor was run using parameters $\gamma = 0.0006$ and $\lambda = 0.2$, which were chosen by empirically testing different values to find the highest ones that did not cause the SGD to diverge. The algorithm was given 3000 iterations to converge while also using the rmsstop criterion. In the end, the algorithm ran a couple of hundred iterations until converging.

The parameters for the SVD with SGD were: $\gamma = 0.0005$, $\lambda = 0.2$ and the number of components was set at 4, as indicated by the results in Section 5.2.2. The algorithm

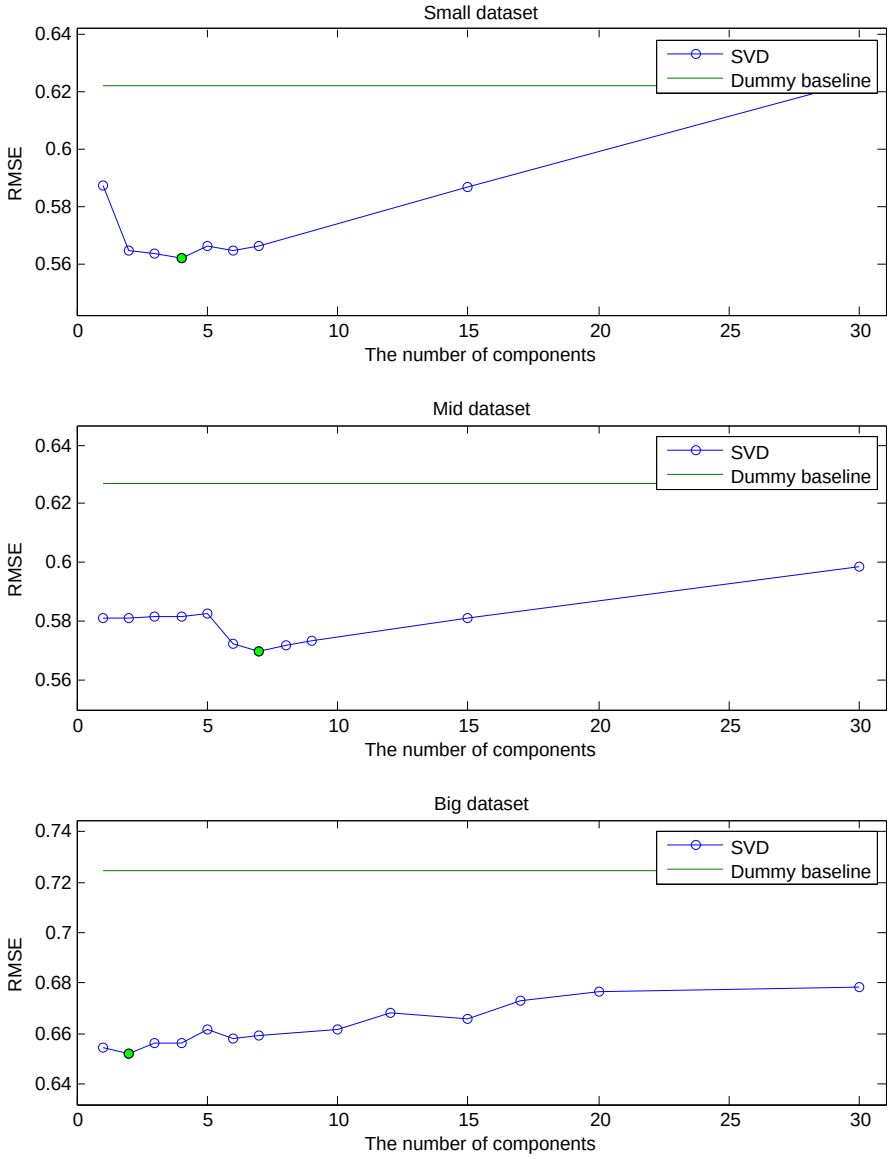

Figure 5.5: SVD performance on all validation sets using a different number of components for the 5-fold cross validation. The lowest RMSE is marked as a green dot. The corresponding lowest RMSEs are achieved using 4 components for small dataset, 7 for the middle dataset and 2 for the big dataset.

was given 5000 iterations with rmsstop. The k-NN model was run with 5 neighbors userwise, 9 linkwise and 21 neighbors using subreddit features.

VBPCA was run with the default settings using the same rmsstop-criterion as other methods with 8 components. Setting the number of components for VBPCA is not so critical since it uses automatic relevance determination (see Section 3.4.3). The number of components was thus chosen to be double the amount of components determined for SVD, since both of SVD and VBPCA are subspace methods. Too few components would make VBPCA underperform and too many would make it overfit the training set leading to poorer performance.

Method	Accuracy	Soft RMSE	Hard RMSE	Class average accuracy	Downvotes	Upvotes
Naive	0.9032	0.6221	0.6221	0.5000	0.0000	**1.0000**
Random	0.8244	0.8381	0.8381	0.4998	0.0972	0.9023
Baseline	0.9161	0.5052	0.5794	0.6183	0.2490	0.9875
SVD	0.9231	0.4875	0.5546	0.6757	0.3688	0.9825
k-NN_SR	0.9176	0.5065	0.5740	0.6282	0.2692	0.9871
k-NN_User	0.9175	0.5058	0.5745	0.6201	0.2513	0.9889
k-NN_Link	0.9222	0.4912	0.5579	0.6545	0.3226	0.9864
VBPCA	0.9243	0.4861	0.5503	**0.6795**	**0.3759**	0.9831
MLRM	**0.9256**	**0.4766**	**0.5457**	0.6710	0.3553	0.9866

Table 5.8: Metrics for the different methods on the small Reddit test set, when the training phase was done on the corresponding testset. Bolded numbers signify the best result for that particular metric.

As expected, the results in Table 5.8 show that MLRM performed best on 3 metrics while VBPCA was very close and was by far the best on estimating downvotes. Much simpler SVD also comes very close to VBPCA in each metric. It is very important to note that both of these single methods outperform the best result that (Poon et al., 2011) were able to achieve using the same dataset. Their best RMSE score was **0.491433** using a linear combination of models while the best score using MLRM in this thesis is **0.4766**, which is **3.11%** percent better. k-NN models perform more poorly than SVD and VBPCA for this dataset and the link-based recommendation approach seems to be the best. The weights of MLRM are presented in Table 5.9, which clearly shows that the highest weights are for the link-based k-NN and VBPCA. Here, the naive method represents the constant term of linear regression. ROC curves for the methods are presented in Figure 5.6, where the MLRM clearly dominates all the others.

5.4 Results for Middle Reddit Dataset

The tests for the mid dataset were run much the same way as for the small dataset. Learning rate for the stochastic gradient descent needed to be tuned down a bit. Thus, the parameters used were $\gamma = 0.00005$ and $\lambda = 0.2$ for baseline predictor and

Method	MLRM weights
Naive	0.0071
Baseline	-0.1869
SVD	0.2656
k-NN_SR	0.1106
k-NN_User	-0.0359
k-NN_Link	0.4615
VBPCA	0.3976

Table 5.9: Weights for the MLRM for the small dataset. Higher absolute values mean higher relevance.

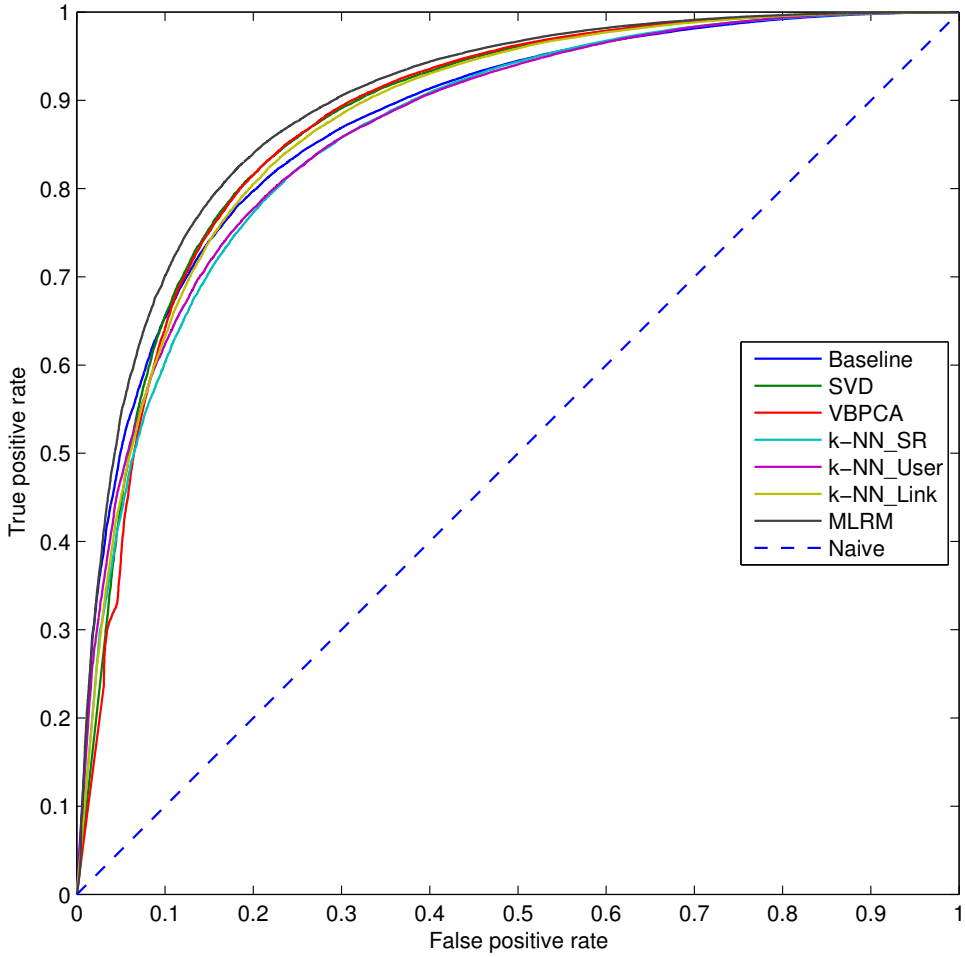

Figure 5.6: ROC curves for the small Reddit dataset.

$\gamma = 0.0001$ and $\lambda = 0.2$ for SVD with number of components being 7. VBPCA was run with the same setting, except changing the number of components to 14, using the same heuristic of doubling the optimal number of components for SVD. The results of the experiments are presented in table 5.10. Surprisingly, link-based k-NN achieves the lowest RMSE of single models and actually performs really close to MLRM. This is no coincidence since its weight for the MLRM is as high as 0.6176 (from Table 5.11), over 50% higher than for VBPCA. However, VBPCA is superior on downvotes metric and thus on class average accuracy also. ROC curves are shown in Figure 5.7.

Method	Accuracy	Soft RMSE	Hard RMSE	Class average accuracy	Downvotes	Upvotes
Naive	0.9018	0.6268	0.6268	0.5000	0.0000	**1.0000**
Random	0.8225	0.8426	0.8426	0.4990	0.0965	0.9016
Baseline	0.9161	0.5038	0.5793	0.6237	0.2599	0.9876
SVD	0.9205	0.4924	0.5641	0.6709	0.3604	0.9815
k-NN_SR	0.9171	0.5058	0.5759	0.6268	0.2655	0.9881
k-NN_User	0.9176	0.5030	0.5740	0.6256	0.2621	0.9890
k-NN_Link	0.9237	0.4833	0.5524	0.6668	0.3470	0.9865
VBPCA	0.9222	0.4879	0.5578	**0.6837**	**0.3870**	0.9805
MLRM	**0.9263**	**0.4715**	**0.5430**	0.6821	0.3782	0.9860

Table 5.10: Metrics for different methods on the mid Reddit dataset.

Method	MLRM weights
Naive	-0.0110
Baseline	-0.3497
SVD	0.1948
k-NN_SR	0.0972
k-NN_User	0.0883
k-NN_Link	0.6176
VBPCA	0.3870

Table 5.11: Weights for MLRM for the mid dataset. Higher absolute values mean higher relevance.

5.5 Results for Big Reddit Dataset

Baseline predictor and SVD were run using parameters $\gamma = 0.00002$ and $\lambda = 0.2$ and SVD was run with two components, as Section 5.2.2 indicated. The same heuristic was used for VBPCA as for the other datasets, thus it was run with 4 components.For some reason the SVD performs poorly. No other model was run with so many different parameters during this thesis but SVD, but still its performance did not improve. VBPCA does still perform very well so subspace models can get good results for this dataset, but for some reason SVD cannot. Linkwise k-NN performs

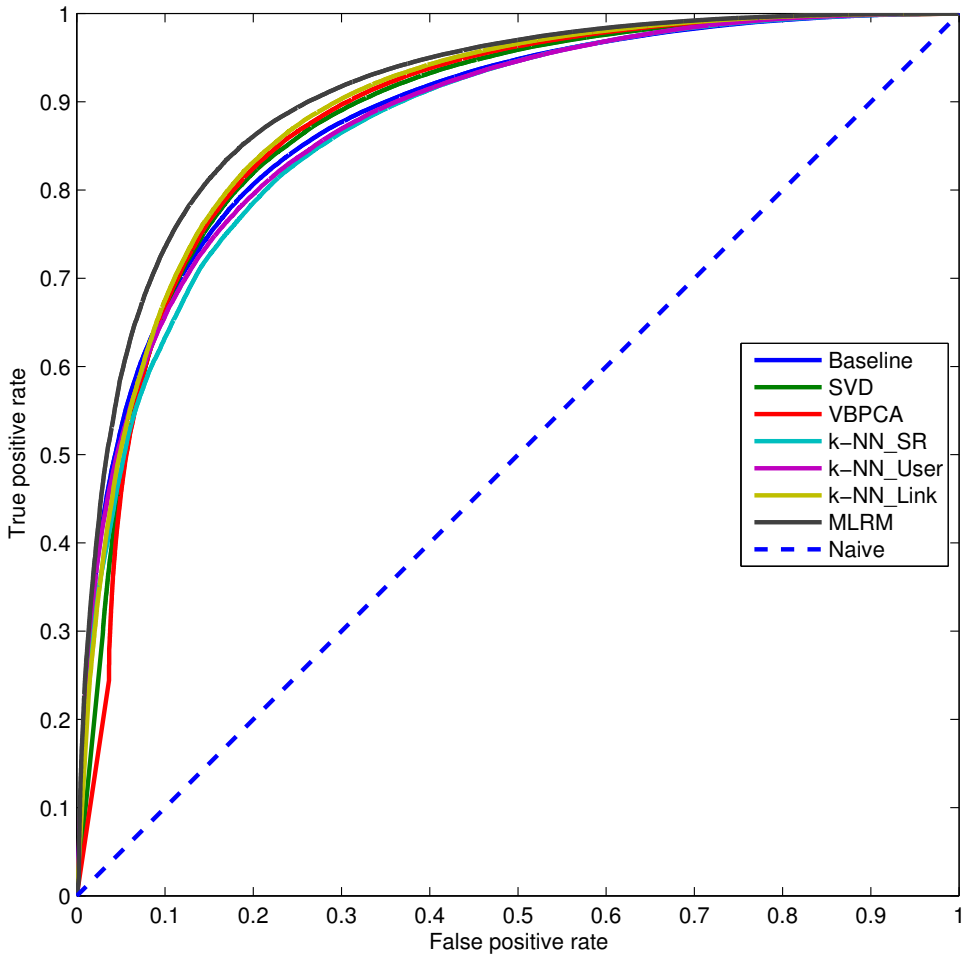

Figure 5.7: ROC curves for the middle Reddit dataset.

very well and amazingly succeeds in estimating almost 45% of downvotes right without compromising the upvote estimation rate. For this reason it also gets the highest weight for the MLRM in Table 5.13. Also, linkwise k-NN and VBPCA combined gets weighted around 90% of the all the methods so leaving all the other methods out from the MLRM model would not decrease its estimation performance significantly. ROC curves in Figure 5.8 show the differences between models clearly. VBPCA and k-NN_Link are above other methods close together while SVD is below the other models and seems like it is underperforming, since its worse than the baseline predictor.

Method	Accuracy	Soft RMSE	Hard RMSE	Class average accuracy	Downvotes	Upvotes
Naive	0.8687	0.7248	0.7248	0.5000	0.0000	**1.0000**
Random	0.7720	0.9549	0.9549	0.5002	0.1316	0.8689
Baseline	0.8937	0.5707	0.6521	0.6630	0.3502	0.9759
SVD	0.8900	0.5859	0.6632	0.6612	0.3508	0.9716
k-NN_SR	0.8942	0.5658	0.6507	0.6682	0.3617	0.9747
k-NN_User	0.8930	0.5697	0.6542	0.6738	0.3766	0.9711
k-NN_Link	0.9048	0.5451	0.6172	**0.7091**	**0.4438**	0.9745
VBPCA	0.8991	0.5505	0.6353	0.6929	0.4132	0.9726
MLRM	**0.9067**	**0.5293**	**0.6108**	0.7043	0.4298	0.9788

Table 5.12: Metrics for different methods on the big Reddit dataset.

Method	MLRM weights
Naive	-0.0150
Baseline	-0.0156
SVD	0.0350
k-NN_SR	0.0032
k-NN_User	0.0942
k-NN_Link	0.5058
VBPCA	0.4075

Table 5.13: Weights for MLRM for the mid dataset. Higher absolute values mean higher relevance.

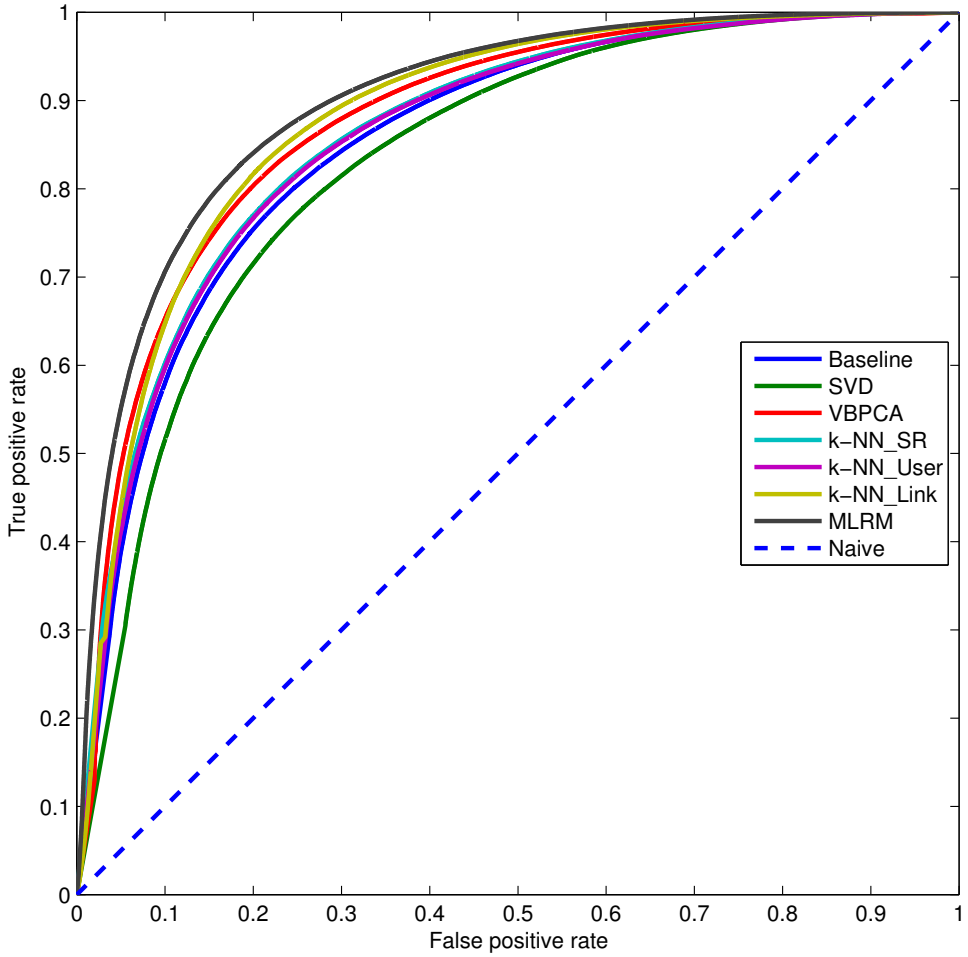

Figure 5.8: ROC curves for the big Reddit dataset.

5.6 Analysis of Results

The results given in sections 5.3-5.5 are mostly for comparing the performance of different methods. To see how many downvotes and upvotes were actually estimated correctly, the confusion matrices (these were explained in Section 4.1) for all datasets are presented in Tables 5.14-5.16. The results are based on the models that performed best on class average accuracy metric, which means VBPCA for the small and mid datasets and k-NN_Link for the big dataset.

		Predicted vote:	
		Upvote	Downvote
Actual vote:	Upvote	196969	13397
	Downvote	3394	8068

Table 5.14: Confusion matrix for the results of VBPCA model on the small dataset.

		Predicted vote:	
		Upvote	Downvote
Actual vote:	Upvote	585266	39850
	Downvote	11632	25155

Table 5.15: Confusion matrix for the results of VBPCA model on the mid dataset.

		Predicted vote:	
		Upvote	Downvote
Actual vote:	Upvote	1621083	139911
	Downvote	42496	111634

Table 5.16: Confusion matrix for the results of linkwise k-NN model on the big dataset.

The rest of this section presents figures representing how dependent the estimation accuracy is on some other factors, such as the upvote ratio of a user. In this context, the estimation accuracy means the estimated accuracy that depends on external factors after the estimation phase is done using the MLRM model for each dataset.

Figure 5.9 shows the upvote ratio compared to estimation accuracy. It seems evident that on average, having a higher upvote ratio increases the estimation accuracy. Since upvotes are generally much easier to estimate than downvotes, this is quite logical consequence of the fact that around 90% of votes are upvotes and thus it is easier to learn the model describing upvotes than downvotes. The increasing trend of the pictures in Figure 5.9 is most obvious with the mid dataset, which also has the highest density of all the datasets in this thesis.

Figure 5.10 presents the number of votes given by users against their estimation accuracy. Since the histogram of given votes is thin tailed (See Figure 5.2), Figure 5.10 also has most of it mass with the users giving less votes.

Figure 5.11 shows how the estimation accuracy depends on the user's activity in different subreddits. The mean curve for the mid dataset seems to have more variation than for the small dataset, but this is highly likely to be a statistical property of the data due to limited number of observations on very low and very high subreddit productivity levels.

The plots in Figure 5.12 are seemingly quite different if the upvote ratio is computed linkwise instead of userwise. This may be because in larger datasets there are more users with lower upvote ratios and since downvotes are more difficult to estimate in general, the trend tends to go downward when upvote ratio decreases.

Figure 5.13 compares the number of votes per link with the estimation accuracy. There is a clear trend that having more votes per link increases the estimation accuracy on average.

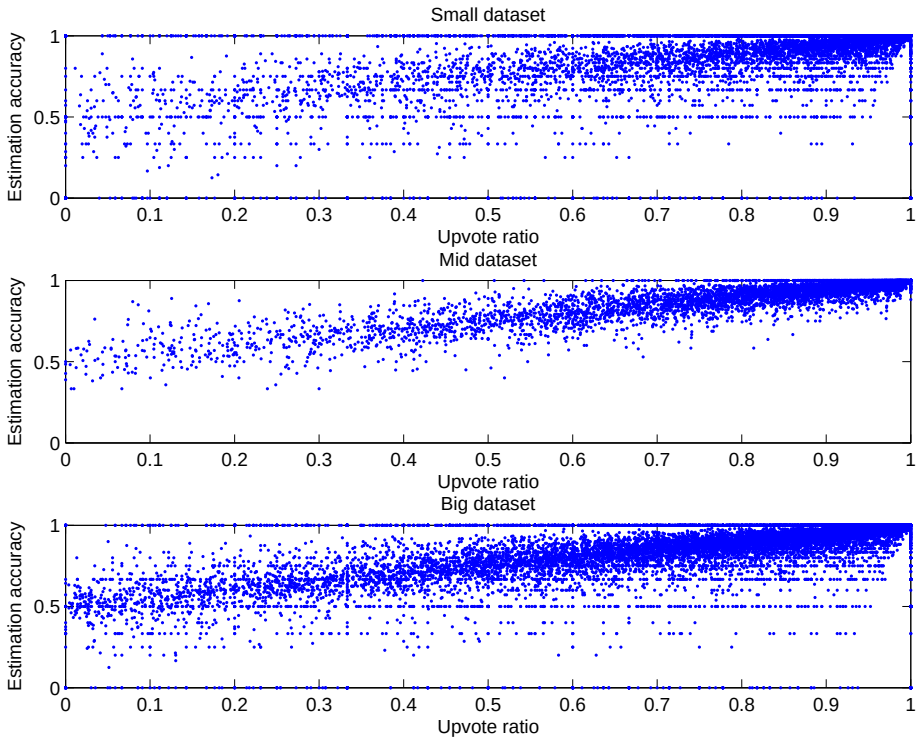

Figure 5.9: Comparing upvote ratio of users with the corresponding estimation accuracy.

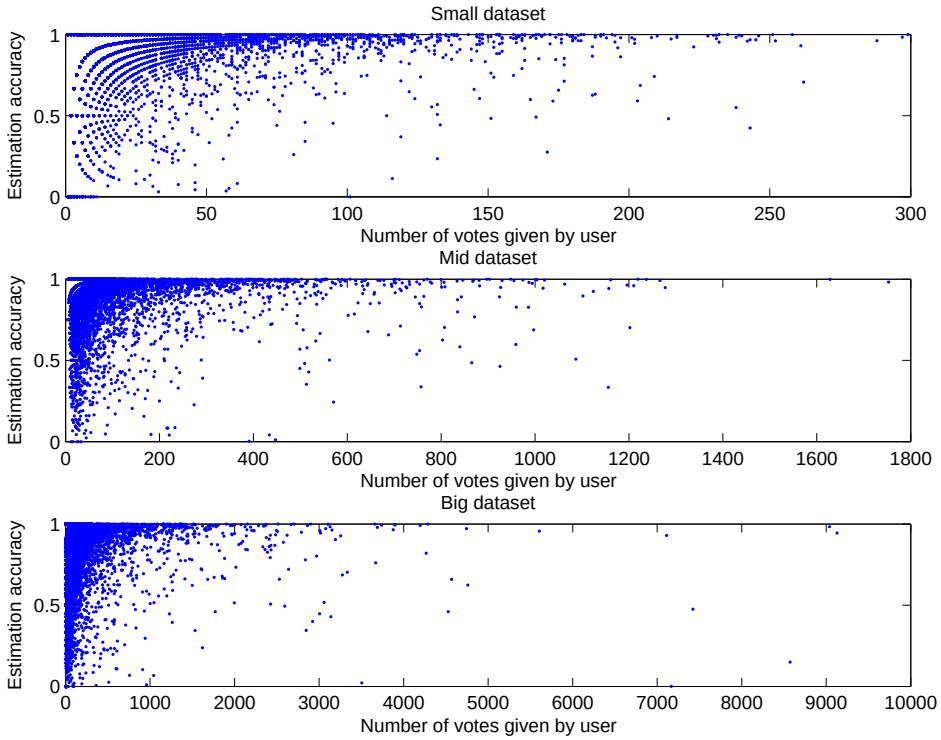

Figure 5.10: Comparing total number of votes of users to their corresponding estimation accuracy. Notice that while the plots seem quite similar, the scaling of horizontal axis multiplies with larger datasets.

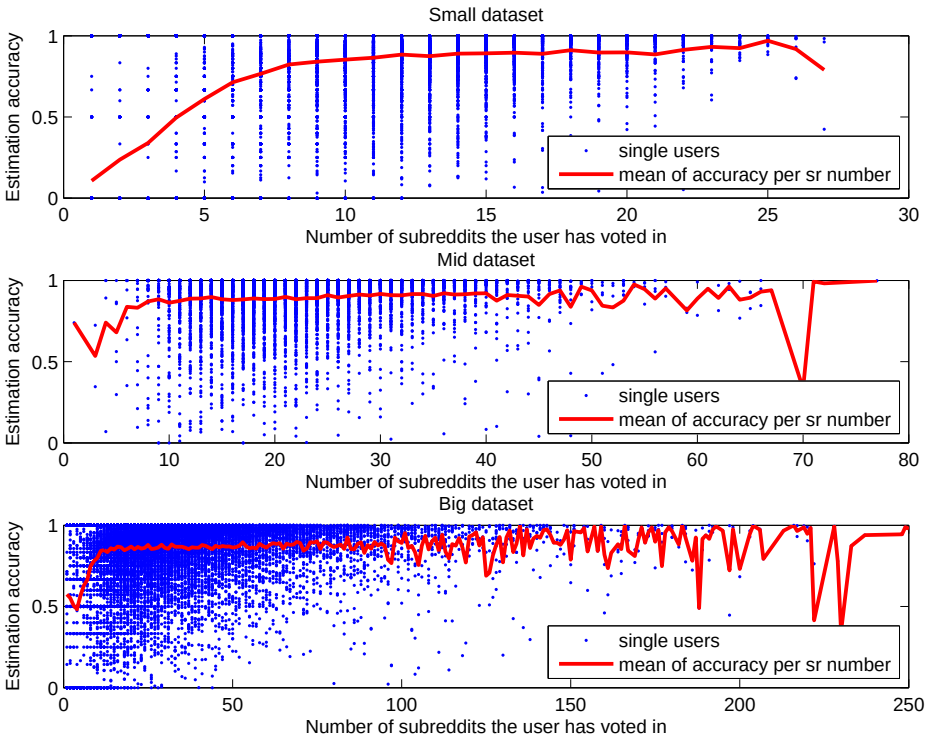

Figure 5.11: Comparing the number of different subreddits where the user has been active to the estimation accuracy.

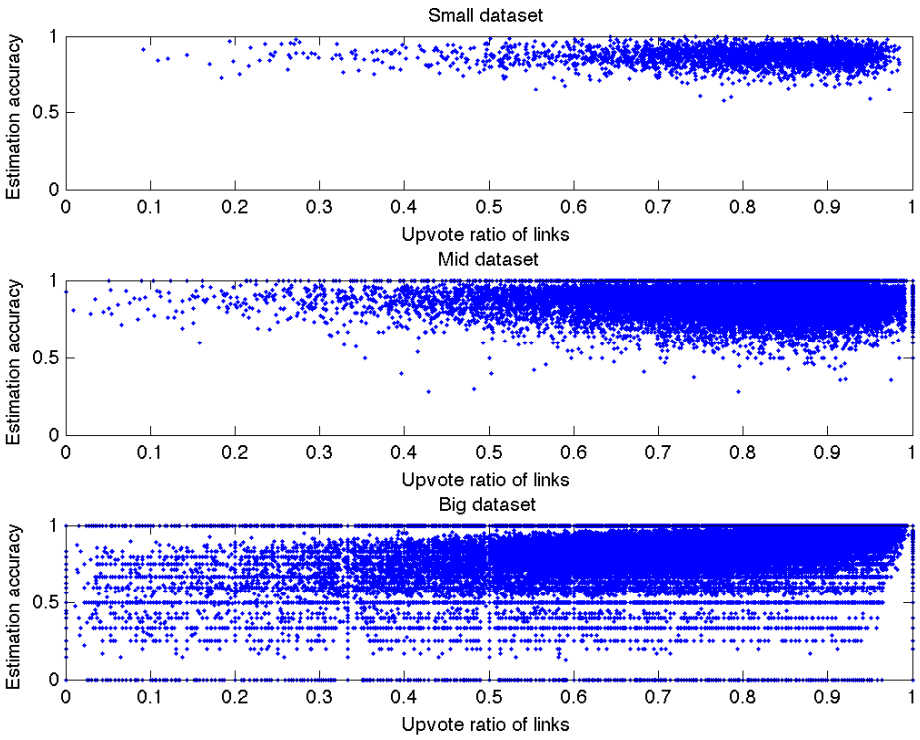

Figure 5.12: Comparing upvote ratio of all links to their corresponding estimation accuracy.

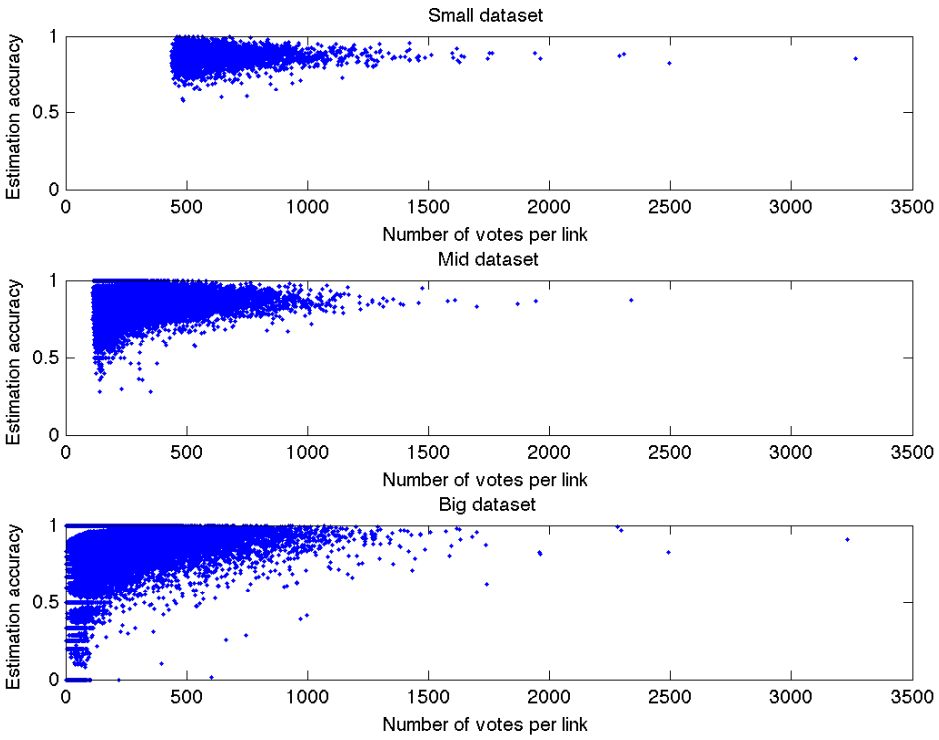

Figure 5.13: Comparing the number of votes per link to the corresponding estimation accuracy.

Chapter 6

Conclusions

In this thesis, the recommendation problem in the context of for or against voting was approached using two kinds of collaborative filtering techniques: neighborhood models and latent factor models. Three ways of computing the k-nearest neighbors were implemented: user-based, link-based and subreddit-based approach. Three latent factor models were also implemented: Baseline classifier, SVD and VBPCA. Baseline and SVD models were implemented using stochastic gradient descent while the VBPCA was applied using the Matlab toolbox developed by Ilin and Raiko (2008). MLRM was used as a simple and robust ensemble method for combining the results.

The Reddit dataset was introduced, analyzed and split into 3 different sized datasets and all the experiments were run on these three datasets with missing-at-random assumption. For finding the best number of neighbors for the k-NN models, 5-fold cross validation was used. The experiments indicated that too many neighbors were better than too few and the optimal number of neighbors was around 20 or more, regardless of how the neighborhood was defined in a particular k-NN model. Choosing more than 20 neighbors did not seem to decrease or increase the RMSE, except for the sr-wise k-NN model.

5-fold cross validation was also implemented for the SVD and the results indicated that the optimal number of neighbors k differed between the datasets. This relation seemed to depend on the size and sparsity of the dataset, such that for the sparser dataset it was more difficult to find relevant components and thus the SVD was implemented with only 2 components for the big dataset, but 7 components for the mid dataset. The number of components for VBPCA was decided to be twice the amount that was estimated for SVD, since VBPCA can use ARD for adjusting the components towards zero, if the evidence of the relevance of the corresponding components is weak.

The results on the small dataset using soft RMSE were shown to be 3.11% better than the best result from Poon et al. (2011). The RMSE from VBPCA, SVD and link-based k-NN were shown to be better than their best result. This was most likely due to implementing SVD with SGD as shown by Funk (2006), VBPCA simply being more advanced model for problems like these and k-NN being implemented in a different way than theirs. VBPCA was also shown to be superior model on downvote estimation for the mid and small dataset, while link-based k-NN was better for the big dataset. The ensemble method MLRM was superior on accuracy and RMSE for all datasets as one would expect, MLRM being the only ensemble method implemented.

From the results of the k-NN classifiers, it is clear that using the user's votes for the most similar links is consistently better than estimating votes given by the most similar users. During the beginning of running the experiments, it seemed like a good idea to use the subreddit features as the distance measure between the users since it reduces the dimensionality and intuitively sounds good. However, this approach did not work as well as the userwise and linkwise k-NN. It is also worth noting that computing the k nearest neighbors based on links or users may easily become intractable, if the user-link matrix becomes less sparse. Even for the user-item matrix being as sparse as it is for the big dataset, the implementation was nontrivial and in actuality probably the most difficult thing to program efficiently during the experiments phase.

In conclusion, VBPCA and link-based k-NN were by far the best models, which is further evidenced by the highest weights on the MLRM model for these two models.

The results could most likely be improved by introducing more classifiers, for example Restricted Boltzmann Machines (Salakhutdinov et al., 2007), or combining the results of several classifiers with some better suited technique, such as stacked generalization (Alpaydin, 2004, p. 364).

The dataset itself could also be improved by additional data, such as including the content of the links to the dataset to enable using content-based approach to this dataset, such as text mining. Another possibility would be to add part of the IP address to the dataset for some general geographical analysis. While this kind of data addition might introduce some privacy or business interest concerns, there are also ways to increase the value of the dataset without introducing any harm to the business or to the users. For example, introducing temporal data to the dataset, e.g., timestamps to all the votes would increase the breadth of models that could be used for the dataset as well as the number of interesting research questions to be asked. One such question would be the extent of multiple user accounts per person for voting, or the usage of throw away accounts for very short term usage. Some cases of such behavior might stand out from the dataset. Also, time aware factor models, such as timeSVD++, have been shown to improve the estimation accuracy on the Netflix dataset (Ricci et al., 2011, p. 160) so it might improve on Reddit dataset too.

From the viewpoint of this thesis, the most useful addition to the dataset would be the "zerovotes", i.e., certainty of a user seeing a link without voting it either way. This would transform the classification problem from 2 class problem to a 3 class problem, where the number of zerovotes would most definitely dominate over upvotes and downvotes.

Hopefully larger and more complete datasets than the Reddit dataset will be released in the future too, for the joy of hobbyists and professionals alike, without compromising the privacy of the users present in the data.

Bibliography

Adomavicius, G. and Tuzhilin, A. (2005). 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.

Alexa. Alexa - reddit.com site info. `http://www.alexa.com/siteinfo/reddit.com`. Accessed January 17, 2013.

Alpaydin, E. (2004). *Introduction to Machine Learning*. The MIT Press, 1st edition.

Bell, R., Koren, Y., and Volinsky, C. (2007). The bellkor solution to the netflix prize. *KorBell Team's Report to Netflix*.

Bennett, J. and Lanning, S. (2007). The netflix prize. In *KDD Cup and Workshop in conjunction with KDD*.

Bishop, C. (1999). Variational principal components. In *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, volume 1, pages 509 –514 vol.1.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Burke, R. (2007). Hybrid web recommender systems. *The adaptive web*, pages 377–408.

Funk, S. (2006). Netflix update: Try this at home (december 2006). `http://sifter.org/simon/journal/20061211.html/`. Accessed January 17, 2013.

Herlocker, J., Konstan, J., Terveen, L., and Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53.

Ilin, A. and Raiko, T. (2008). Practical approaches to principal component analysis in the presence of missing values. `http://users.ics.aalto.fi/alexilin/papers/tkk-ics-r6.pdf`.

Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer, second edition.

King, D. (2010). Want to help reddit build a recommender? – a public dump of voting data that our users have donated for research. `http://redd.it/dtg4j`. Accessed January 17, 2013.

Koren, Y. (2008). 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.

Lagus, K. Virtualcoach blog. `http://blog.pathsofwellbeing.com`. Accessed November 24, 2012.

Mahmood, T. and Ricci, F. (2009). Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 73–82. ACM.

McSherry, F. and Mironov, I. (2009). Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 627–636, New York, NY, USA. ACM.

Moore, D. S. and McCabe, G. P. (2006). *Introduction to the Practice of Statistics*. W. H. Freeman and Company, 5th edition.

Németh, B. and Tikk, D. (2007). Major components of the gravity recommendation system. *ACM SIGKDD Explorations Newsletter*, 9:80.

Netflix. Netflix company timeline. `https://signup.netflix.com/MediaCenter/Timeline`. Accessed January 17, 2013.

Netflix (2009). Netflix prize webpage. `http://www.netflixprize.com/`. Accessed January 5, 2012.

Noeva, P. (2012). Sampling methods for missing value reconstruction. Master's thesis, Aalto University School of Science.

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, volume 2007, pages 5–8.

Poon, D., Wu, Y., and Zhang, D. Q. (2011). Reddit recommendation system. `http://cs229.stanford.edu/proj2011/PoonWuZhang-RedditRecommendationSystem.pdf`. Accessed January 17, 2013.

Reddit. reddit: the front page of the internet. `http://www.reddit.com/about/`. Accessed January 17, 2013.

Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors (2011). *Recommender Systems Handbook*. Springer.

Rubin, D. B. (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *ACM international conference proceeding series*, volume 227, pages 791–798.

Salihefendic, A. (2010). How reddit ranking algorithms work. `http://amix.dk/blog/post/19588`. Accessed January 17, 2013.

Schwartz, B. (2004). *The paradox of choice: Why less is more*. Harper Perennial.

Seidman, S. (1983). Network structure and minimum degree. *Social networks*, 5(3):269–287.

Siltanen, S. and Müller, J. L. (2012). *Linear and Nonlinear Inverse Problems with Practical Applications*. Society for Industrial and Applied Mathematics.

StackExchange. Free, community-powered Q&A. `http://www.stackexchange.com/`. Accessed January 17, 2013.

Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2.

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2008). Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 267–274. ACM.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622.

Vatanen, T. (2012). Missing value imputation using subspace methods with applications on survey data. Master's thesis, Aalto University School of Science.