Recommender Systems and the Netflix prize

by

Simen Gan Schweder

THESIS

for the degree of

MASTER OF SCIENCE

UNIVERSITY OF OSLO

Faculty of Mathematics and Natural Sciences
University of Oslo

November 2008

Det matematisk-naturvitenskapelige fakultet
Universitetet i Oslo

Contents

1 Preface

My interest in recommender systems started in 2000, when my good friend Øystein M. Gutu suggested creating a movie recommender for films being displayed at movie theathers in Oslo. We consulted a friend studying statistics at the University of Oslo, Håvard Broberg, and created the Dirichlet estimation described in section 3.4.1, this worked fairly well and substantial improvement turned out to be dificult. Later on another post doc student Pierpaolo De Blasi suggested a neighbourhood model that never was realized. For about a year Oslo Kinematografer, the owner of most theathers in Oslo, paid to have our recommender engine on their site, but new managment failed to see the value and cut our funding. Due to bad research at the time, we thought we were the only living beings on the planet interrested in these matters. Then in 2006 the Netflix competition, introduced in section 5, came along and shattered all illusions of uniqueness.

My father, Tore Schweder, has some responsibility for making me interested in statistics, as this is also his profession. But I was not particularly inclined to study the subject until I took some mandatory courses in statistics taught by Nils Lid Hjort. He made the subject look easy whithout hiding the details, and showed the power of modern statistics in a convincing way. I am very thankful that Nils later accepted to be my supervisor on this Master project. Many bad cups of coffee and good conversations have been very inspiring.

My girlfriend Venke Uglenes and daughter Ada has shown great forgiveness for me turning half our living room into an office and ranting on about movies and ratings over the last year.

Thanks also to Alfaprint AS, my current employer who have made this possible.

2 Introduction

Since the introduction of the Internet, the amount of information available to those connected has exploded. Prior to this, information was stored internally (in our heads), written in books, on local harddisks or to be found in other peoples internal storage. Different methods of querying were used for the different medias.

Information stored in one's own internal storage was readily available and retreived by largely unknown techniques.

Information stored in books was searched by first finding the right book, either by internal knowledge or browsing through bookshelves, and then examining the list of content or an index.

Information stored on one's harddisk could perhaps be searched by keywords, but often this had to be browsed through manually.

Information stored in other people's internal storage was queried by vocal or slow textual queries in an informal language, often leading to erroneous searches.

As of Februrary 2007, an informal estimate of the total number of webpages available to all of us (the connected ones), is about 30 billion pages. A large number, although a low proportion, of theese provide information from wast databases. The total amount of available information is baffling.

A natural question is then how to order this amount of information?

The librarians have studied a version of this problem for ages. The Library of Alexandria, built early in the third century BC, set on fire by Ceasar 48 BC, crippled by the cristian Emperor Theodosius I, and finally destroyed by the Muslim conquest 643 AD, had a collection of works estimated to be between a few hundred thousand to a millon documents. This was the cradle of the scientific method, scholars from "all over the world" took recidence here to study the scrolls. The organization of the libraries was done by cataloging. Second hand sources suggest that the lead librarian, Kallimachos created the catalog Pinakes, wich consisted of no less than 120 books. Pinakes was titled 'Overview of the most prominent men in all branches of science and their written work', and contained an alphabetic list of authors with information about place of birth, teachers, parents etc. It also contained an index of different versions of each document. The documents themselves were organised by classes like medicine, history, poetry, etc. [8]

Melvil Dewey published his Dewey Decimal Classification as a system for organizing books in library shelves in 1876. The system was originally based on ten main classes which in turn are divided into ten sections that are further divided in ten divisions. Hence the system consists of 10 classes, 100 sections and

1000 divisions. Each book is then assigned a tuple of numbers separated by the decimal separator ".". The first number consists of one digit for the class, one for the section and one for the division. This thesis could perhaps be classified as 006.31, where "006" consists of "General Texts/General/Programming", and ".31" is "Artificial Intelligence/Machine Learning".

Further on one can add numbers to identify geographic regions, time or cross reference sections. If one should try to classify the pages found on the World Wide Web in this system, one would end up with an average of 30 million pages in each category, and still the underlying information available from many of the pages would make them extremely difficult to classify. On top of that, one would have to be a librarian to be able to search the reservoir of information available.

In the period between the birth of the Internet and the birth of the World Wide Web the main thing to search for was filenames. Personally my first searches on the internet were done by a service named Archie, to perform a search I sent a email containing my search terms, and a few minutes/hours later the search results, filenames matching the query, was returned by mail. Very impressive at the time (1990?).

After the birth of the World Wide Web, in CERN, 1991, the search for information changed. The introduction of HTML, a standard way of representing formated text/pages, along with the first web-browsers lowered the bar for publishing and browsing the information available to the selected few that were connected to the web. Now the first spiders started to crawl the web, constructing indexes of keywords and mapping out the graph. Several major search engines emerged. One of them, Excite (1993), was the first to use statistical analysis of word relationships to automatically classify pages. Yahoo! was founded in 1994, and quickly became a leading search engine, or rather a directory of webpages. Yahoo! both created human made compilations of webpages, and organized the pages in a Dewey-like hierachy. Google emerged in 1998, and their PageRank algorithm for ranking search results (discussed later) quikly put them in the lead of the search marked.

With all this sophisticated search technology one would think it should be easy to find what one is looking for. And it is! That is, if you know what you are looking for and what you are looking for can be phrased easily. If you are wondering what 'light' is, enter 'light' as your search term in Google, and it will return more than 900 million pages, but do not dispair, they are sorted by the magical pagerank algorithm, the first one is the Wikipedia entry on 'light'. If you want to know the variance in the gamma distribution, you can find that

equally easy. It is a little more difficult to find out how to do things, like how to change the breakpads on a bike, if you enter something like 'how to change breaks on bikes', the top search entries are a mixed bag, half of them are directly relevant, the others not at all.

The real problem starts when you want to find something specific to you! How about finding a book you would want to read? Or a place you want to live? What color to paint your living room? This is information that does not exist, no one has published a webpage with the required information, and probably no one will ever do just that. With some effort you could probably find a good book by reading lots of reviews, or browsing top lists from others and comparing their tastes in books to your own. This process is one of the topics researchers attempt to automate with the use of Recommender Systems.

This thesis will introduce the reader to Recommender Systems in the next section, including some examples from different kinds of systems. In section 4 I introduce some standard implementations of Recommender Systems including Matrix Factorization by Singular Value Decomposition and the K-nearest neighbours method. The Netflix Prize is introduced in section 5 along with a short discussion of its strengths and shortcommings. Some of the entries to the Netflix Prize are reviewed in section 6, and my own three implementations are covered in section 7, including the novell logarithmic model. We take a look at the difficulties in sampling from such a interconnected set as the netflix movie dataset as presented by Art B. Owens in section 8. Finally some concluding remarks and a preview of the road ahead is presented in section 9.

3 Introduction to Recommender Systems

Recommender systems are a subclass of Information Filtering systems.

> An Information filtering system is a system that removes redundant or unwanted information from an information stream using (semi)automated or computerized methods prior to presentation to a human user. Its main goal is the management of the information overload and increment of the semantic signal-to-noise ratio. [18]

A recommender system tries to evaluate the interest/taste that a specific user has to a set of items. E.g. the web search engine, Google, searches through an enormous amount of web pages when you enter your search term, and then attempts to rate those pages with respect to what is most interesting, and presenting the search result sorted by your anticipated interest in the particular page. The "Pagerank"-algorithm used by Google to perform this evaluation is regarded as one of the most important pieces in building the company that has grown to be among the the world's most successful.

There exists an enormous need for recommender systems to filter the vast amounts of information available on the Internet, or items available in for instance online shops, and therefore this is a rapidly growing industry. It is used widely in large online solutions, here are some examples:

- Google as mentioned above.
- Amazon uses it to recommend books when the customer is in the shop, and to send personalized recommendations by email.
- iRead recommends books inside the popular Facebook framework.
- GroupLens on USENET Net news, a world wide internet discussion/news system, recommends articles to users.
- Netflix, a US based dvd-rental company, uses a recommender system to let customers find movies they would like.
- Match.com, a matchmaking site, matches users by profiles compiled by questtionaires.

Formally a recommender system can be formulated as follows: Let C be the set of all users, and I be the set of all items that can be recommended, R is a totally ordered set(eg. reals in $[0, 1]$). Let $u : C \times I \to R$ be the utility function that measures the utility of item $i \in I$ for user $c \in C$. So our problem then is to find this utility function.

Recommender systems come in two basic flavours, content based systems and collaboration filtering systems, these can be mixed and blended as will be shown below. There are also two distrinct theoretical implementational appraches to recommender systems, a model based approach and a heuristic approach. We will introduce the different flavours and approaches next.

3.1 Content based systems

In a content based recommender system the utility function generally recommends items similar to what the user has preferred in the past. Generally this is based on

- $F_I(i)$, a vector of the features of item i, e.g. author, title, number of pages, publisher, genre of a book.
- $F_C(c)$, a vector of the features of user c, often a history of ratings from the user, historical behaviour or a user profile compiled from a questionnaire or simply a search term.

The utility function is then on the form $u(c,i) = score(F_C(c), F_I(i))$.

3.1.1 Example: term frequency/inverse document frequency (TF-IDF)

This is a tool picked from the librarian's toolchest, it generates a profile for a document consisting of a vector of weights based on keywords. The underlying assumption is that the relative frequency of the selected keywords characterize the documents.

Assume we have N documents and that keyword k_j appears in n_j of them, and let f_{ij} be the number of times keyword k_j appears in document d_i. We define the Term frequency (normalized frequency) TF_{ij} of keyword k_j in document d_i as:

$$TF_{ij} = \frac{f_{ij}}{\max_z f_{iz}}$$

where $\max_z f_{iz}$ is calculated over all keywords that appear in documend d_i. Further we define the Inverse document frequency for keyword k_j as

$$IDF_j = \log \frac{N}{n_j}.$$

Finally we compute the TF-IDF weight for keyword k_j in document d_i as

$$w_{ij} = TF_{ij} \times IDF_j.$$

The content of a document is then, $content(d_i) = (w_{1i}, ..., w_{Ki})$, where K is the number of keywords.

After this ordeal we have placed each document in a K-dimensional space and can use the cosine between the documents as a similarity measure. In a search context one uses the cosine between the search text and the documents as a measure of fit. A profile of a user can consist of selected keywords or a list of previously viewed documents. See [13] for a textbook introduction.

3.2 Collaboration Filtering systems

> Collaborative Filtering, the sharing of knowledge through recommendations.

The Collaboration Filtering (CF) approach to recommender systems takes advantage of other user's input to the system. The general idea of CF is that people that have agreed in the past tend to agree in the near future as well. One can go further in this direction and try to decompose the opinions of users on different items to model how users evaluate items. CF techniques will be the main focus of this thesis and ample examples will be given.

3.3 The Wisdom of Crowds

The idea of a Collaboration Filtering System can be traced back to Francis Galton (1822-1911), a Brithish scientist who was also a half cousine of Charles Darwin and a Fellow of the Royal Society, knighted in 1909. On one occasion our Francis wisited the "West of England Fat Stock and Poultry Exhibition" as some of his many interrests was the measurement of physical and mental qualities and breeding. On the Exhibition cite, a weight-judging competition was arranged. A large ox was on display, and for a six-pence one could by a ticket where the competitors entered their name and guess to the slaugther weight of the ox. The competitors consisted of a fair mix of "experts", farmers and butchers, and "non-experts", clerks and others with no apparent expert knowledge. Galton, who was an extreme elitist, had no faith in the guesses of the non experts, and thus the mean outcome of the competition. The mix of experts and non-experts led him to belive the average outcome would miss the mark by a mile. So after the competition he borrowed the tickets and did a little study on them. The slaughter weight of the ox turned out to be 1198 pounds, and the average of the guesses, there were 787 of them, was 1197 pounds. Galton himself viewed the average as the collective wisdom of the crowd, and the near perfect result greatly increased his belief in democracy which he viewed as another aspect

of The Wisdom of Crowds. Or in his own words: "The result seems more creditable to the trustworthiness of a democratic judgement than might have been expected". The quote is taken from the book "The Wisdom of Crowds" [15], whitch describes several interesting experiments that seem to show that the collective choices of individuals often are remarkably good. The general idea here is that the individuals themself have small fractions of knowledge about the problem at hand, and use their knowledge and understanding of the domain to make predictions. When aggregated the errors cancel out and what is left is a destilled result from all the fractions of knowledge. Several experiments indicate that under the right conditions the average of a crowd's guesses to some problem is normaly closer than 98% of the participants guesses, even if the crowd is a mixed lot with varying knowledge of the problem. Of the most prominent conditions for this to work is that the participants do not communicate while making their decissions. The author actually claims that many decissions in businesses fail because of over-communication in the board.

3.3.1 Example: Google's Pagerank algorithm

When searching for a term in google the results are returned sorted by their so-called Pagerank. The Pagerank attempts to estimate how important or relevant each page is to the search term provided. The method used in ranking the pages is often coined as The Democracy Of The Web. This method ranks a page by the number of other pages that link to it, and the pagerank of those other pages. More formally the Pagerank is calculated as follows:

$$PR(A) = (1-d) + d\sum_{i=1}^{N} PR(T_i)/C(T_i),$$

where N is the number of pages, $T_1, ..., T_N$ pointing to A, $d \in [0,1]$ is a damping factor (usually set to 0.85), C(T) is the number of links going out from page T. For further explanation see the article by Sergey Brin and Lawrence Page introducing Google [5]. This algorithm is straightforward, and does not use any user profiles.

3.3.2 Example: K-Nearest Neighbours

The "standard" implementation is a k-nearest neighbours approach, so if we want to make a recommendation for a specific user, we first find his neighbours, those that have agreed with the user in the past, and then see what the neighbours think of the item in question. The KNN approach will be discussed further in section 4.2 and in section 6.1.5.

3.4 Hybrids

Very often the final recommendation system actually used by companies are hybrids of Content Based and Collaboration Filtering systems. One often tries to implement item-item similarities by inspecting both the known facts of the item (genre, language, actors etc. for films) and the users' recommendations of the item, see section 6.3 for an example. User-user similarities are on the other hand most often based solely on their recommendations.

3.4.1 Example: Dirichlet estimation in eSmak

A Norwegian movie portal, eSmak.no, was created by Øystein Michael Gutu and the author. The users are here asked to give ratings to different movies and then a "guestimate" of their opinions on other movies are calculated.

We keep a lot of information about each movie, including title, original title, actors, director, language, country of origin. We also classify each movie in one or more genres, e.g. a movie can be classified as action, drama or action/drama. All information is automatically ripped from the homepages of Oslo Kinomatografer. Sadly our cooperation with Oslo Kinomatografer has come to an end, and our automatic ripper is now blocked from their site. While the cooperation was still on, we classified 949 movies, there where 2000 unique users and 18900 ratings collected. We also automatically ripped the ratings of critics in all major newspapers. One early idea was to match each user to newspaper critics, and extrapolate the users ratings based on how well they matched the different critics. The advantage of this is that the critics have rated close to all movies, giving a good basis for estimation. However this method is nicely enveloped by the K-Neares Neighbours approach.

The algorithm we ended up using is a simple hybrid of a Collaboration filter and a Content based recommendation system that uses the genre of the movies and user ratings as its only content. The profiles of the users are the rating distribution for different genres. Ratings in this system is from 1 to 6, where 6 is best.

The idea is to see the choice of a rating as 6 rivaling events, corresponding to each possible rating, described by a Dirichlet distribution. We approach this in a Bayesian manner, letting the prior distribution be $(X_1, ..., X_6) \sim Dir(\alpha_1, ..., \alpha_6)$, where X_i is the probability of giving the movie a rating i, and α_i is the number of ratings with the value i the movie has received. We update our distribution with information about how the user has rated movies in the genere in the past. So the user has its own histogram, $(\gamma_1^{ug}, ..., \gamma_6^{ug})$, where γ_i^{ug} is the number of ratings with the value i the user, u, has given in the genere g. To

balance the distributions, we also weight them to match importance down to their variances, giving a posteriori distribution of $(X_1^*, ..., X_6^*) \sim Dir(\beta)$, where $\beta_i = \frac{w_\alpha \alpha_i}{\sigma_\alpha^2} \sum r_{u\bullet} + \frac{w_\gamma \gamma_i}{\sigma_{ug}^2} \sum r_{\bullet m}$, and $\sum r_{u\bullet}$ is the number of ratings given by the user, and $\sum r_{\bullet m}$ is the number of ratings received by the movie. The w_α and w_γ, are used to controle how much one relies on the users rating history vs. the movies rating history. Currently they are both set to 1.

We predict the rating $\hat{r}_{um}$, for user, u, and movie, m, of genere g as the expectation of X^*. The expectation of X_i when $X \sim Dir(\beta)$ is simply $E(X_i) = \frac{\beta_i}{\sum_j \beta_j}$. So our $\hat{r}_{um} = \sum_{i=1}^{6} i \frac{\beta_i}{\sum_{j=1}^{6} \beta_j}$. In the final implementation this guestimates turned out to be overly conservative. A recommender system that gives very conservative recommendations feels irrelevant by the user. We sincerly believe that strong opinions, eaven when not to well founded, are more interresting to the user than more well founded weak opinions, or plainly stated, its better to have an opinion than not have anything to say. We fixed this by streching our guestimates with a sigmoid function, putting more weight in the ends of the scale. The guestimates presented to the user was actually $\hat{r}_{um}^* = C_0 + \frac{C_1}{1+e^{-C_2(\hat{r}_{um}-3.5)}}$, where the constants, C_0, C_1, C_2 where chosen by hand. This gives fairly radical guestimates, while still seperating between movies we think the user will like or dislike.

3.5 Blending different approaches

Finally the most efficient systems often come from blending different soulutions to the problem. As an example, the long time leader of the Netflix Prize competition (introduced in section 6.1) blended a full 107 different systems to reach their annual progress prize winning score. What seems to be the mainstream approach here is to use a linear regression on the predictions of the different soulutions using the target ratings as the response. They further emphesize that diversity in the different soulutions seems more important than refining a single soulution.

Another approach not much used in this field is Bagging and Boosting. Bagging would, in this context, work by creating an ensamble of predictors by retraining our recommender system on resamples of the original dataset, then a voting scheme is implemented where each of the trained predictors has one vote each. Studies on Bagging [4] suggest that they can be usefull if the learning algorithm is unstable, whitch is not the case for most of the algorithms presented in this thesis. A problem with this method is the resampling, it is shown in section 8 that resampling from datasets on the form used in this thesis is not as easy as one would think.

Boosting consist of methods to create strong prediction rules based on a combination of weak/inaccurate/rough prediction rules, or plainly phrased, combine several rules of tumb to a strong prediction rule [14]. The best known algorithm for Boosting is AdaBoost for classification problems and AdaBoostR for regression problems. Further resources on Boosting can be found on
http://www.boosting.org/

3.6 Model based approaches

In a model based approach we attempt to fit a statistical model to the data and use the model to predict recommendations. The clear advantage of this way of doing things is that we get a lot of free theory from the field of statistics, like variance analysis, confidence intervals and hypothesis testing. Also, creating a statistical model gives a rationale for the result, we can understand the model and results with a statistical foundation. The example given in section 3.4.1 is clearly model based, this can be extended using standard statistical theory to predict the joint rating of two or more users on a given film simply by creating a joint distribution from the individual users and using its expected value as a prediction. Or, more elaborately, we can search for the film that has the largest 5% quantile for the users, thereby "ensuring" that the experience will be positive for all. Other typical implementations are logistic regression where qualities of items are treated as categorical data and linear regression. We will also take a look at a certain logarithmic model in section 7.2.

3.7 Heuristic approaches

In a Heuristic approach we do not create an explicit statistical model for the data, but rather attempt to create a simplified representation that to a largest possible extent gives a predictive ability.

Typically this approach consists of defining

- some parameterized prediction function: $PREDICT(\text{User } u, \text{Item } i; \theta)$
- an error function: $ERROR(\theta)$, witch is often defined as the sum of the squared prediction error, $ERROR(\theta) = \sum_u \sum_i (PREDICT(u, i; \theta) - r_{ui})^2$ where r_{ui} is known.

Then find some mimimization algorithm that can minimize the error function and thereby finding $\hat{\theta} = argmin_\theta$. The prediction will then yield a single number for a given user and item with $\hat{\theta}$. This approach does not directly give us any statistic on the prediction, we will need to use external techniques to find the

variance of the prediction. We will later introduce the Factorial Decomposition (often a SVD) as an example on heuristical approaches (section 4.1.5). We will also look into some bootstrapping techniques to aggregate statistics from the results (section 8).

4 Implementations of Recomender Systems

In this section an overview of the standard implementations of recommender systems is introduced. There is a supprisingly diverse flora of implementations, with very different approaches. Some are based on traditional statistical models, others are very inventive in their assumptions. We will present the most common implementations including Matrix Factorization by Singular Value Decomposition, K-Nearest Neighbours and Restricted Boltzman machine. Typical implementations not presented here include Clustering, Neural Nets and more.

Another important part of the implementation is selecting the data to work with. One normal way of organizing the data is to have a user-item matrix with some sort of score in each cell, but the origin of this information can be diverse. In the Netflix dataset we typically use the rating of each movie as the cell-value, Amazon.com on the other hand could use a value of 1 if the user bought the book, and 0 otherwise. Google represent their data as a graph where the individual internet pages are nodes and the hyperlinks are the edges connecting those nodes.

The evaluation of the recommender system also deserves some attention. We usually evaluate a recommender system by splitting the data in two parts, a training set and a probe set. We then train our system on the training set, and evaluate it on our probe set. In the Netflix Prize competition, the measure to use is defined in the rules as Root Mean Squared Error (rmse), where the error is the difference between the predicted rating and the actual rating. iRead has a different measure of quality. Although their algorithms are not public, they have reviled some details. Their measure of quality is based on wheter a user "takes positive actions on recommendations", meaning that the user clicks on a suggested book and thereby confirming at least a superficial interrest in the recommended book. This decoupling introduces another challenge in the learning algorithm, as the error function is not directly a function of the parameters itself, but rather a secondary effect.

4.1 Singular Value Decomposition

We will look into how Singular Value Decomposition (SVD) can be used as a heuristic implementation of a recommender system, often as a collaboration filter. We include the needed theorem from linear algebra and discuss how it can be used in this context.

4.1.1 Theorem

For the real case the Singular Value Decomposition(SVD) Theorem [19] ensures that we can decompose a real $m * n$ matrix A as follows:

$$A = U\Sigma V^t$$

Here

- U is a $m \times m$ orthonormal matrix
- Σ is diagonal $m \times n$ matrix with the singular values of A on the diagonal.
- V is a $n \times n$ orthonormal matrix

A singular value is defined as any real non-negative σ such that there exists unitvectors $\bar{u} \in \Re^m$ and $\bar{v} \in \Re^n$ and $A\bar{v} = \sigma\bar{u}$ and $A^t\bar{u} = \sigma\bar{v}$.

4.1.2 Approximating matrices

An interesting application of SVD is in approximating the matrix A by a matrix $\tilde{A}$ such that $rank(\tilde{A}) = r < rank(A)$. We can define such an approximation by minimizing the Frobenius norm $||B||_F^2 = \sum_1^m \sum_1^n |b_{ij}|$ where $B = A - \tilde{A}$ restricted to $rank(\tilde{A}) = r$ where r is given.

It turns out that the optimal solution for this problem is given by the SVD of A as follows:

$$\tilde{A} = U\tilde{\Sigma}V^t$$

where U and V is as before and $\tilde{\Sigma}$ is the same as Σ above except that only the r largest σ's are retained, the rest is set to zero.

4.1.3 Missing values

The theory above breaks down when the matrix in question has missing values. In our application of SVD we work with matrices where most of the entries are missing, for example the unusually rich (in this context) Netflix dataset has about 99% missing values. So we need an algorithm to deal with such sparse, in the meaning of incomplete, matrices. One standard way of dealing with missing

values is by imputation, e.g let the missing values be the row-means or column-means and then apply the theory above. This can be useful when the number of missing values is relatively small, but makes no sense with the sparsity we are to work with.

Let us rephrase the problem as trying to estimate a matrix A with the product of two matrices U^*, V^* such that $A \approx U^*V^*$ and $rank(U^*V^*) < rank(A)$. Using the Frobenius norm as a measure we want to minimize $||A - U^*V^*||_F$ restricted to $rank(U^*V^*) = r$.

Note that we can use the theory above even when using only two matrices to estimate A by writing $U^* = U\sqrt{\Sigma}$ and $V^* = \sqrt{\Sigma}V^t$, giving $U^*V^* = (U\sqrt{\Sigma})(\sqrt{\Sigma}V^t) = U\Sigma V^t$. Here U^* is an $r \times m$ matrix and V^* is an $n \times r$ matrix.

4.1.4 EM approach

Presented here is a sketch of an iterative algorithm for a Expectation Maximation (EM) approach to finding the decomposition.

Start by fixing U^* to some value $\hat{U}$, we then estimate V^* by the $\hat{V}$ that minimize $||A - \hat{U}\hat{V}||$ this can be found by setting

$$\hat{V}^t = (\hat{U}^t\hat{U})^{-1}\hat{U}^tA \tag{1}$$

then estimate $\hat{U}$ by minimizing with respect to $\hat{U}$, and viewing $\hat{V}$ as fixed.

$$\hat{U} = A\hat{V}(\hat{V}^t\hat{V})^{-1}. \tag{2}$$

Repeat (1) and (2) until convergence.

For a discussion of this algorithm see [11], or [2, (page 99)]. Althoug not apparent, this method requiers imputations on the matrices to perform the calculations above, specifically this is necessary in the matrix inversions. In the method described by Roweis [11], maximum likelihood estimation is used for imputations, but these imputations cause serious problems when the sparsity is as high as in our problem domain.

4.1.5 Gradient descent approach

The idea behind gradient decent is to create an error function $e(A, U, V)$ that takes the matrices A, U, V as its parameters and returns the error $A - UV$ as its value. Then we attempt to minimize the square of the error function by following the gradient of the error function with respect to U and V. We find the partial derivates of e^2 with respect to U and V and use the result to correct

our estimates of U and V. More precisely: Let the estimate (or prediction) be $P = UV$, the error is then $e = A - P$, and the partial derivates

$$\frac{\partial e^2}{\partial U} = 2e\frac{\partial e}{\partial U} = -2e\frac{\partial P}{\partial U}, \tag{3}$$

$$\frac{\partial e^2}{\partial V} = 2e\frac{\partial e}{\partial V} = -2e\frac{\partial P}{\partial V}. \tag{4}$$

We then update our U and V by moving them down the slope given by the partial derivates. Note that gradient decent is only guaranteed to work when the error function is convex and sufficiantly smooth, we can see that each gradient is linear, thereby staisfying this criterion.

A specific implementation of this method is presented in sections 6.2 and 7.1.

4.2 K-Nearest Neighbours

The most common implementation of a Recommender System is by a K-Nearest Neighbours algorithm. In the early days the dominating method used user-user similarities to estimate a users liking of an item. Given a similarity measure s_{ij}, denoting the similarities between user i and j, an estimate of user i's rating of item v could be calculated as

$$\hat{r}_{iv} = \frac{\sum_{j \in M(v)} s_{ij} r_{jv}}{\sum_{j \in M(v)} s_{ij}}, \tag{5}$$

where $M(v)$ is the set of users to previously rate item v.

This looks very intuitive, but one piece of the puzzle is missing, namely the similarity measure s_{ij}. How do we measure the similarities of two users by the available information. If we are working with a movie recommender system, the information we have on a user could be restricted to the previous ratings of this user. So the similarities between two users is down to the similarities between two vectors of ratings. This vectors probably have lots of missing values, that is movies not yet rated by the users. The most common way of measuring similarity between users is the Pearson correlation on the common support of the users, that is the tendency for users to rate items similary, and is expressed as

$$s_{ij} = \hat{\rho}_{ij} = \frac{\sum_{v \in V} r_{iv} r_{jv} - n \bar{r}_i \bar{r}_j}{\sqrt{n \sum_{v \in V} r_{iv}^2 - (\sum_{v \in V} r_{iv})^2} \sqrt{n \sum_{v \in V} r_{jv}^2 - (\sum_{v \in V} r_{jv})^2}},$$

where V is the set of items rated by both i and j, and $n = |V|$.

The algorithm name, K-Nearest Neighbours, suggest that we should not take all other users into account, but only look at the K nearest ones as measured by s_{ij}. We arrange this by introducing $M_i^K(v)$, the K-nearest neighbours to user i with known ratings for v. And substituting $M(v)$ with $M_i^K(v)$ in equation 5.

Later on item-item similarities became popular. In this version we estimate the rating r_{ik} by looking at how user i rated items similar to item k. So the similarity measure is between items instead of users. s_{kl} denotes the similarity between items k and l and is computed in the same way as above except the sums are over users that have rated both items. Item-item similarities play a large role in online shopping, where the site whishes to recommend items to users based on what they have already bought or placed in their shopping basket. E.g. we would like to recommend a hammer to a customer who has bought nails. If previous users who has bought nails also bought hammers the similarity between nails and hammers would be high, and thereby giving us the information needed to make the recommendation. And since the similarities

are symmetric, a recommendation to buy nails when buying a hammer is also feasable.

There are some fundamental problem with using KNN as a predictor in the domain of movie ratings. It is very hard to predict that a user would give a rating of 1 to a specific movie. This is because the neighbours of the user probably has not seen the movie, they know its bad, so the closest neighbours of the user that have seen the movie is probably far away. And in an item-item settings, the user has probably not seen any movies like the one in question. Another problem stems from the fact that the KNN method is basicaly a weighted mean, the prediction always ends up somewhere in the center of gravity between you neighbours ratings, making the method conservative.

There are many ways to improve the accuracy of the K-Nearest Neighbours method, including shrinking of similarities based on the size of the common support, combining item-item similarities with user-user similarities and others. Koren and Bell discusses a model approach to Neighbourhood predicting in [1], this is presented closer in section 6.1.5.

4.3 Restricted Boltzmann Machine

A Restricted Boltzmann Machine is a stochastic neural net consiting of two layers, hidden nodes and visible nodes, see figure 2. There are symmetric connections between every pair of hidden and visible nodes, but not between hidden and hidden or visible and visible nodes. The state of each node is stocastic and depends on its weighted input, that is the sum of all connected nodes multiplied by the connection weights.

The basic task of a RBM can be seen as learning the distribution of a set of input patterns. This is done by a learning algorithm called Contrastive Divergence Learning whitch is a variant of the gradient ascent method.

So, let our network consist of visible nodes $V_1, ..., V_n$, and hidden nodes $H_1, ..., H_m$, these are connected by a symmetric matrix of weights $W = \{w_{ij}\}$ connecting visible node i to hidden node j. Our training set consist of a set $\{t_k\}$, binary vectors of length n.

The input, z_i to a visible node V_i is calculated as the weighted sum of the hidden nodes, $z_i = b_i + \sum_{j=1}^{m} w_{ij} H_j$ where b_i is the bias of the node. The activation s_i of a visible node V_i is stocastic with probability $p(s_i = 1) = \frac{1}{1+e^{-z_i}}$ For the hidden nodes, the values are calculated in the same manner.

Reconstructing a pattern is done as follows:

1. Set the values of the input nodes to t_i.
2. Calculate the values of each hidden node j.
3. Calculate the values of each visible node i.
4. Repeat steps 2 and 3 a number of times.

After these steps, the values of the output nodes are called a reconstruction.

To train a RBM, we do the following for each t_k:

1. Set the values of the input nodes to t_i.
2. Calculate the values of each hidden node j.
3. Let S^1 consist of values $s^1_{ij} = s_i s_j$.
4. Do a reconstruction as outlined above.
5. Let S^n be calculated as in step 3, but with the reconstructed state.
6. Calculate the Contrastive Divergence (CD) as $S^n - S^1$. This is an approximation to the gradient.

7. Update the weights, $w_{ij}^{new} = w_{ij}^{old} + \alpha CD_{ij}$, where α is the learning rate.

The presented RBM has only binary nodes. We will see how this can be used to model ratings later on in section 6.4, but for now lets look at a simpler task namely to model whitch movies are seen by each user. We could model this using an RBM by letting each visible node represent a movie, and adding a suitable number of hidden nodes, preferrably much lover than the number of visible nodes. We then train the RBM as outlined above. To make predictions as to whitch movies a new user would want to see, we set the values of the input nodes for the movies we know the user has seen, and do a reconstrucion.

5 Netflix Prize

5.1 Presentation

The Netflix Prize was launched in 2007 on "http://www.netflixprize.com/". Its goal is to increase the prediction strength of its own movie recommendation system *Cinematch*. They boast a one million $US award to the first team who increases on the accuracy of their own system by 10%.

> The Netflix Prize seeks to substantially improve the accuracy of predictions about how much someone is going to love a movie based on their movie preferences.

Netflix has released a free (requires registration) dataset consisting of 100 million dated ratings collected in their system over the past 3 years, and the titles for all movies rated. And a qualifying set without ratings, an entry to the contest is made up of predictions for this qualifiing set.

A rating in the dataset consists of:

- movie-id, identifying a single movie.
- rating, that is a number from 1 to 5 where 5 is best.
- user-id, identifying a single user.
- date, the date of the rating.

An item in the qualifying set consists of the same items except of course the rating.

The competition has been a huge success, several teams are competing for the prize. The teams consist of researchers, students, freelance phsycologists, engineers in garages, and others. One of the reasons for the success is that Netflix mandates that every winning algorithm must be published in full, including a paper that explains the reasoning behind the algorithm. This has lead to several well written articles and web-pages, see especially [2], [17].

5.2 Wrong question?

Are Netflix asking the wrong question? The Netflix competition has released a training set of 100M ratings, and asks the competitors to predict a qualifing set of about 10K users. The naive approach would be to build a model reflecting how well each user likes each movie, and using this to predict the qualifiing set directly. But we would then miss one important implicit information bit,

Figure 1: Image from the Netflix Prize

namely the ratings in the qualifiing set is actual ratings, meaning the persons who rated them actually choose to watch that movie. On the other hand, the competitors only work with ratings that are actually made, and the problem thereby naturally rectifies itself in terms of the competition, but the goal of the competition is slightly skewed.

The Netflix competition proposes to ask the question: How well would person A like movie B? But rather asks the question: How well would person A like movie B provided that he choose to watch it.

The question of most interrest would be: How well would person A like movie B? This is the information that would allow us to suggest movies, compile top-lists of unseen movies and provide the most usefull information to the users.

But this information is not easyli available from data sets only containing ratings on volunteerly seen movies, at least when no extra data on the movies are available. A better suited dataset for this study could consist of test subjects set to watch random movies and then rating them, this would of course be much more resource dependant than merely asking for ratings on movies volunteerly seen. Other proposed methods of modelling this question would be to add more data on the movies, and for instance create a regression scheme on the covariance matrix of these extra pieces of data. On could image adding data like genere, actor, publishing year, director etc., and use methods from multivariate statistics to model the preferences of users with respect to the covariates. This would of course still be conditioned on the user having seen the movie volenteerily, but

one might think that the information provided by the model would still be valid down to some intercept for involunteerly seen movies. The latest entries to the Netflix Prize (not covered furteher in this thesis) have started to use implicit information in their modelling, the favourite simply beeing a boolean user times movie matrix containing 1's where the user have rated the movie and 0's otherwhise, then modelling their data conditioned on this matrix.

6 A look at the entries to the Netflix Prize

The Netflix Prize continously maintains a leaderboard for the competition. Here the names of the leading teams are presented as well as their best achivements in terms of rmse. The leaderboard can be found on http://www.netflixprize.com//leaderboard. As mentioned, the main prize is one million dollars to the team that first breaks the 10% improvement barrier on Netflix's own Cinematch recommender system. There are also an annual progres prize, fifty thousand dollars are given to the team that has the best score each year. So far two progress prizes have been awarded, the fist one won by a team named KorBell, their effort will be introduced in section 6.1. The second progress prize was awarded BellKor (the same participants as KorBell) in collaboration with BigChaos. The rules of the Netflix Prize states that each progress prize winner as well as the grand prize winner have to publish their work, giving students, like myself, and researchers an exelent source of information.

Several of the non-winning teams also publish their methods in more or less formal ways. An early leader on the leader board, Simon Funk, described an SVD approach outlined in section 6.2. The team Gravity has held a top-10 position for a year or so, their approach is also a SVD like approach, but they add some static elements to their analysis. Some of their specifics are presented in section 6.3.

6.1 Korbell

6.1.1 The team

The Korbell/Bellkor team have three primary contributors, Yehuda Koren, Bob Bell and Chris Volinsky, all from AT&T research. They have been active in the competition from the very start and has an impressive suite of algorithms at hand. They are also among the few that publish their work with serious foundation in statistics. Currently they are at the very top of the leaderboard, and have won the two only progress prizes awarded, the last in collaboration with a team named BigChaos.

6.1.2 Shrinking estimated parameters

Shrinkage is viewed as a continous alternative to parameter selection in the methods used by the Korbell team. They view shrinkage as a result from the Bayesian point of view where estimated parameters are viewed as data. The shrunk parameter is then calculated as the posterior mean, a linear combination of the prior and the estimated parameter. E.g. if we are to estimate a column mean (movie mean), $\bar{m}_0$, we could see that as drawn from a prior distribution with mean equal to the global mean of all movies, $\bar{m}_{total}$. This would give a shrunk estimate of the movie mean

$$\bar{m}_{shrunk} = \bar{m}_0 + \frac{var(\bar{m}_0)}{var(\bar{m}_0) + \frac{1}{n}var(\bar{m}_{total})}(\bar{m}_{total} - \bar{m}_0).$$

Also in their models shrinkage is applied on estimated parameters, the general method is to estimate a parameter through normal machine learning, shrink it towards zero, remove the predicted effect, and then go on to train the next parameter on the resulting residuals.

6.1.3 Removal of global effects

They always remove the most obvious global effects from the data before trying more sophisticated modelling later on. The global effects are modeled as $b_{im} = b_0 + b_i + b_m$, where b_0 is the average rating over all movies, b_i is the offset of movie i, and b_u is the offset of user u. The values for b_i and b_u are a bit harder to come by than one would think. The naive estimation of e.g. $b_i = \frac{1}{N_i}\sum_v r_{iv} - b_0$, where N_i is the number of ratings for movie i and r_{iv} is the rating of movie i by user v where it exist, does not distinguish between effects from users and movies. So we need a way to simultauosly solve b_i and b_u One way of solving

this would be to minimize the following:

$$\min_b^* \sum_{(i,u)\in K} (r_{iu} - b_0 - b_i - b_u)^2,$$

but this leads to overfitting and needs regularization. The Korbell team solves this by instead minimizing the following:

$$\min_b^* \sum_{(i,u)\in K} (r_{iu} - b_0 - b_i - b_u)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2),$$

where the added terms penalize the magnitude of the parameters.

Also a regression on the ratings vs. the time stamps, to remove the time-dependant effect is sometimes applied.

6.1.4 Neighbourhood based estimation - standard

First a little notation: i,j denotes movies, u, v denotes users, the set $N(i;u)$ denotes the set of movies that is rated by user u, and $M(u;i)$ denotes the users that have rated movie i.

As seen in section 4.2 the standard way of applying knn is a user based neighbourhood estimation where you basically look at your neighbours and see what they like. A neighbour is someone who has similar taste as yourself. If we are to predict the rating of user u on movie i, r_{ui}, we first build the set $M(v;i)$ consisting of all users, v, that have rated movie i. Then let the prediction be

$$\hat{r}_{ui} = \frac{\sum_{v\in M(v;i)} s_{uv} r_{vi}}{\sum_{v\in M(v;i)} s_{uv}},$$

whitch is simply a weighted mean of the neighbour's ratings. Of course we need to have some similarity measure for this method to work. Of the methods mentioned in the article is the Pearson correlation coefficient or the Cosine similarity.

Another option is a movie based neighbourhood estimation, where you look at the movies your previously have rated and their similarity to the movie in question. So we generate the set $N(i;u)$ consisting of all movies, i, you, u, have seen, then predict your rating for movie i, r_{ui} in the following manner:

$$\hat{r}_{ui} = \frac{\sum_{j\in N(i;u)} s_{ij} r_{ui}}{\sum_{j\in N(i;u)} s_{ij}},$$

that is, the weighted average of the movies you have rated where the weights are the similarities to the movie in question.

Two problems were noted:

- There is no real justification for the similarity functions, the choice to use e.g. a Cosine similarity function does not have any real-world or model based foundation.
- There are no considerations of item-item correlations. They use the example of the Lord of the Rings trilogy, the three films are probably very equal in most similarity functions, and thus will count three times in any prediction, where they should probably be regarded as just one rating.

6.1.5 Neighbourhood based estimation - Korbell

Instead of using a generic similarity function to weight the neighbours the Korbell team suggests a more model-like approach. We switch notation from similarities, s_{ij}, to weights, w_{ij} to emphesize this. Instead of a predetermined similarity function we now search for the set of weights that gives the best predictions. The following notation is used in the expression below: i is the movie to be rated, $U(i)$ is the set of all users who have rated i, $N(i;u,v)$ is the set of movies rated by both user u and user v. We solve the least squares problem:

$$\min_w \sum_{v \in U(i)} \left(r_{vi} - \frac{\sum_{j \in N(i;u,v)} w_{ij} r_{vj}}{\sum_{v \in N(i;m,n)} w_{ij}} \right)^2 .$$

A problem with the expression above is that it does not consider how many films the user v has seen, neither does it consider if the ratings of v is similar to the ratings of u. The authors claim to remedy this by weighting user v with $c_i = \left(\sum_{j \in N(i;u,v)} w_{ij} \right)^\alpha$, with $\alpha = 2$ we get the complete expression:

$$\min_w \sum_{v \in U(i)} c_i \left(r_{vi} - \frac{\sum_{j \in N(i;u,v)} w_{ij} r_{vj}}{\sum_{v \in N(i;m,n)} w_{ij}} \right)^2 / \sum_{v \in U(i)} c_i .$$

When the weights are constrained to be non-negative, the equations are no longer linear, and needs more complicated methods to solve. The Korbell team uses a version of gradient projection [9] to find their weights.

6.1.6 Neighbourhood aware retraining of SVD

An interesting idea by Bell, Koren and Volinsky [2, pages 101-102], is to re-train the local user features of our SVD once we know which movie-user rating we want to predict.

Let us say we want to predict the rating for user u and movie m, r_{um}, and we have user features U and movie feature M, normaly this would give prediction $\hat{r}_{um} = U_u M_m^t$. We can however utilize our knowledge of what prediction we

are to make by re-traing the userfeatures to emphasize the neighbourhood of the movie m. Lets assume we have a similarity measument s_{ij} giving a value for the similarity of movies i and j. We could then re-train our userfeatures by minimizing $\sum s_{mj}(r_{uj} - U_u^* M_j^t)^2$ where the sums are over all j where the rating r_{uj} is known.

The re-training is done quickly given that we have the relevant similarities, s_{mj}, and the actual ratings, r_{uj}, available. We use the following algorithm to find our neighbourhood aware userfeatures:

NeighbourhoodUserFactor(Ratings: r_{ij}, user u, movie m, moviefactors M)
    % Initialize residuals.
    for each rating r_{ui}
        res $\leftarrow r_{ui}$

    for l=1,...,f
        $p_u^i[l] \leftarrow \frac{\sum s_{ij} res_j M_{jl}}{\sum s_{ij} M_{jl}}$
        for each rating $r_u j$
            $res \leftarrow res_j - p_u^i[l] M_{jl}$

Note that the ratings are double centered(movie means and user offsets are removed), so the residuals are simply initialized by the ratings.

In the above, we assume a similarity function on movies, $s(i,j) \rightarrow \Re$ such that similar movies get a high score, and diverse movies score low. The distancefunction used seems to be the inverse average sqared distance between the ratings of the two movies, $s_{ij} = \frac{|M(i;w) \cap M(j;w)|}{\sum_{v \in M(i;w) \cap M(j;w)} \sqrt{(r_{vi} - r_{vj})^2}}$. This is according to a forum response from Mr. Koren on http://www.netflixprize.com//community/viewtopic.php?id=920.

> Inverse average squared distance between ratings on the two movies' common support (similar to what BellKor describes as an alternative to Pearson in section 4.1 of "Improved Neighborhood-Based Collaborative Filtering")

We have however experimented with a more directly spatial arrangment of movies and users in our "Metric Neighbourhood Predictor" implementation. It would be interesting to use $s(i,j)$ as the Euclidian distance in this arrangement instead, thereby decoupling the distance between movies and the re-training of user features.

6.2 Simon Funk - Try this at home

6.2.1 SVD-like implementation

Simon Funk [17] was early on in the competition high up on the leaderboard, and has been very forthcoming with describing his efforts on a personal blog. Here he explains how he uses gradient descent to arrive at the f-rank SVD-like approximation to the real rating set. In the many attempts to improve the prediction accuracy of his algorithm he removes himself from the underlying theory of SVD, but as long as the only goal is to increase prediction accuracy on this specific set of data it should not matter.

The implementation uses a gradient descent approach, that he amazingly is trying to protect with a patent. See section 4.1.5 for an introduction to the gradient decent approach.

We repeat the partial derivates of the error function from section 4.1 here:

$$\frac{\partial e^2}{\partial U} = 2e\frac{\partial e}{\partial U} = -2e\frac{\partial P}{\partial U}, \tag{6}$$

$$\frac{\partial e^2}{\partial V} = 2e\frac{\partial e}{\partial V} = -2e\frac{\partial P}{\partial V}. \tag{7}$$

A *feature* is here defined as the contribution of the k'th singular vector to the approximation. To train the first feature, that is the the rank 1 SVD approximation, we attempt to minimize $||R - U_1V_1^t||$ where U_1 and V_1^t are rank 1 matrixes(row and collum vectors), and R is the true ratings. So we use gradient descent as follows:

- Initialize the U_1 and V_1 matrixes, in theory the choosen values does not matter since the problem does not have any local minima. This can be seen from observing that the partial derivates are all linear.
- Calculate the prediction-error, $E = R - U_1V_1^t$.
- Update U_1 by setting $U_1 = U_1 + \delta EV_1^t$ where δ is the learning rate, set to some value including the 2 from the equation 6 above. Note that we add the error, since we want to travel down the slope. The slope has a negative sign as can be seen in equation 6.
- Update V_1 by setting $V_1^t = V_1^t + \delta EU_1$.

Repeat until convergence witch is guaranteed by the fact that only a global minima exists.
We then repeat the following process to train the $f - 1$ next features, thereby building U_f and V_f matrixes to approximate R with rank f.

- Initialize a new row vector U^* and append this to the matrix U_{f-1} and a row vector V^* appended to V_{f-1}.
- Calculate the prediction error $E = R - U_f V_f^t$.
- Update U^* by setting $U^* = U^* + \delta E V^{*t}$.
- Update V^* by setting $V^{*t} = V^{*t} + \delta E U^*$.

By the fact that convergence to the global minimum is guaranteed the algorithm does indeed produce the true SVD approximation of rank f to R, and thereby the optimal soulution constrained to the form(linear combinations), rank and measured by the Frobenius norm witch ignores missing values.

See the peronal blog of Simon Funk[17], entries "Netflix Update: Try This at Home" and "Netflix SVD Derivation", where this approach is described partially in c-syntax.

Several steps are taken to improve the prediction accuracy, they are presented next.

6.2.2 Shrinkage

The idea behind shrinkage is to impose a penalty on those parameters with little suport in the data. This can be justified from a Bayesian view, we can understand the column means (movie average) as drawn from some underlying prior distribution with a mean equal to the global mean, and calculate the posterior distribution regarding the column mean as data. This would give an expectation proportinal to the ratios of variance between the column mean and the global mean.

The naive column mean for column i looks like this $\bar{a}_{(i)} = \frac{1}{R_i}\sum_j a_{ij}$, where R_i is the number of rows containing data in column i, and a missing a_{ij} is assumed to be 0 in the summation.

The expectation of the posteriori distribution is a blend of the column mean and the global mean,

$$\bar{a}^*_{(i)} = \frac{\bar{a}\frac{\sigma^2_{(i)}}{\sigma^2_a} + \sum_j a_{ij}}{\frac{\sigma^2_{(i)}}{\sigma^2_a} + R_i}$$

where $\bar{a}$ is the global mean, $\sigma^2_{(i)}$ is the variance in column i, and σ^2_a is the total variance.

While tuning the algorithm, Simon Funk reports that it turned out to be a good approximation to fix $\frac{\sigma^2_{(i)}}{\sigma^2_a}$ to the value 25, and thereby avoiding the calculations of variances alltogether.

6.3 Gravity R & D

6.3.1 The team

The Gravity team comes from a team of Budapest University Phd's. They run a company that specializes in recommender systems. They have generously published the main pieces of their implementation in the article [16].

6.3.2 Matrix Factorization

Also the Gravity team has a strong belief in the use of Matrix Factorization(MF), they use the basicly same form as Simon Funk's SVD approach. They do however have some modifications that derserve some comment.

They introduce constant values into their movie feature matrix indicating the existence of certain keywords in the movie titles. Specifically they mention appending a constant column in the movie feature matrix where a 1 indicates the word "season", indicating that the movie is infact a series, and 0 indicating the lack of "season". This column is of course not affected by the learning algorithm, so updates of the movie features is only applied to the columns precieding the constant values, the update of the user feature matrix however does include the movie constants.

Gravity also emphasize the importance of building several as distinct as possible matrix factorizations as possible, and use a blend of these as the final result. They achieve diversity by parameterizing the MF process, with parameters including:

- The number of features.
- Learning rate and regularization factors.
- Distributions for initialization.
- Offset of the rating matrix.
- Nonlinear functions applied to the output.

They do not discuss the effects these parameters have on the result. Should it be suprising if the initialization of the matrixes has an effect? Not really, if the matrix we were trying to factorize was complete the result should be the same every time, the gradient descent algorithm does indeed produce the unique SVD of the matrix. However our rating matrix is not complete, meaning that factorizations are not unique, thus the predictions created by different factorizations could vary.

6.4 ML@UToronto

This team from the university of Toronto have focused on the use of Restricted Boltzmann Machines in their entry to the competition. The team introduces the notion of a "softmax" unit to represent the different values a rating can have, that is a vector of length 5, where each value represents the probability that the user gives a rating of that magnitude. The most interresting aspect of the teams efforts is the way they handle missing data. If all users had seen all movies, we would have a RBM with one "softmax" unit for each movie, and then treating each user as a training case. The hidden nodes would then learn the dependencies between the different ratings. However, we have lots of missing data, and the solution is to create a RBM for each user, where the visible nodes correspond to the movies actually seen by that user, see figure 2. So how then can this machine learn from other users? The trick is to share the weights among all the users, so if two users have a couple of movies in common (they have both seen them), the weights between the hidden nodes and the visible nodes corresponding to these movies are the same for the two users. This also means that each RBM only has a single training case, but the weights still have many.

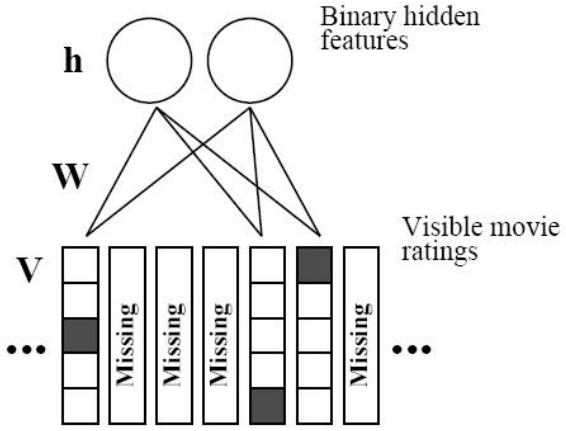

Figure 2: Restricted Boltzmann Machine

The "softmax" units each have a distribution conditioned on the hidden nodes, the team uses a conditional multinomial distribution given as:

$$p(v_i^k = 1|h) = \frac{\exp(b_i^k + \sum_{j=1}^{F} h_j w_{ij}^k)}{\sum_{l=1}^{K} \exp(b_i^l + \sum_{j=1}^{F} h_j w_{ij}^l)}, \tag{8}$$

where v_i^k is i'th component in the softmax unit corresponding to movie k, b_i^k is its bias, h_j is a hidden node, and w_{ij}^l is the weight between then i'th component

in the k'th softmaxunit and the hidden node. And a conditional Bernoulli distribution for the hidden nodes:

$$p(h_j = 1|V) = \sigma(b_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k w_{ij}^k), \tag{9}$$

where h_j is a hidden node, V is the vector of visible nodes, b_j is the bias of the hidden node and σ is the logistic (or sigmoid) function, $\sigma(x) = \frac{1}{1+e^{-x}}$.

In the paper presenting their work, [12], they also discuss strategies to maximizing the likelihood function, and argue that it can not be computed in less than exponential time, making it untractable. They also raise the possibility of using Markov Chain Monte Carlo thechniques for estimation, but concludes that they are to slow and their results vary to much. The authors then fall back on the known technique of Contrastive Divergence, using Gibbs samples from equations 8 and 9. This then has the form:

$$\Delta W_{ij}^k = \epsilon(< v_i^k h_j >_{data} - < v_i^k h_j >_T), \tag{10}$$

where $< \cdot >_T$ represents the sampled distribution initialized at *data*, running for T steps.

A very interesting thought is brought forward when discussing Conditional RMB's, here they touch upon the theme of inferring information from missing data, and thereby indirectly on the issue of Netflix asking the wrong question as briefly discussed in section 5.2.

The idea here is to subtract an amount w from each of the weights connecting hidden units to the softmax units and adding w to the bias. Since the softmax model is overparameterized, all five possible outcomes are modeled, this does not have an effect of a trained machine as long as there are atleast one of the weights between the hidden units and one of the softmax units present. However it does have an effect if none of the connections to the softmax units are present, meaning the rating is missing. This would produce an effect of $-w$ on the input to the hidden unit, thereby including information on missing data in the learning.

The team also find a nice way to "cheat" when it comes to predict the qualifying dataset provided by netflix. For users in the qualifying set with more than one rating they actually use the information about which other movies the user have seen. Even if the ratings are unknwon, the fact that they have seen these other movies provide usefull.

Finally the team describes how to use matrix-factorisation on the parameter matrix of the RBM to reduce the dimensionality of their optimalization problems. This, they claim, leads to much faster convergence.

7 My implementations

I have implemented a few of the methods mentioned above, and two novell algorithms to predict ratings in a user-move scenario. My SVD-implementation is presented in section 7.1, and is a basic implementation of the ideas by Simon Funk and others. The two novell algorithms are a rather naive Metric Neighbourhood Predictor, and a more sophisticated Logaritmic modell.

Training models on data sets as large as the Netflix data set is challenging. The original data set consists of 17703 files, one for each of the 17700 movies with their ratings, one with the movie names, a probe set consisting of about 1.4 million ratings that are also included in the movie files, and finally the qualifying set consisting of about 10 thousand movie-user pairs that are to be predicted as an entry to the competition.

Most of my implementation is done in Java. Traditionally Java is seen as a slow language in terms of running speed, however new virtual engines that compile the code to machine language on the fly results in Java programs running only about 20% slower than hand-crafted C-code.

Several preprosessing steps have been taken.

- It is essential to remove the ratings of the probe set from the training set, not doing so will lead to unfounded optimism as I soon discovered.
- Reassign all movie id's, so they are a sequence from 0 to the number of movies. This allows us to build matrices using movie id's as an index.
- Reassign all user id's, so they are a sequence from 0 to the number of users. This allows us to build matrices using user id's as an index.
- Calculate and store all movie and user means, this greatly speeds up the runtime.
- Compress the data, this reduces the time to read the ratings in the training set from 10 minutes to 2 minutes.

The library written to do the preprosession and modelling is available on request. It has a few main parts worth outlining.

- A File handler, capable of reading the training and probe set. Also able to create histograms and cdf's of ratings for users and movies.
- A Rating Set, representing a set of ratings, can be sorted on either user id or movie id.

- SVD, a representation of a Singular Value Decomposition, capable of serializing/deserializing to/from file.
- Metric Neighbourhood Predictor, capable of learning from a rating set.
- A vizualisation of the Metric Neighbourhood Predictor.
- Lots of functions to extract data from different models and rating sets.

7.1 SVD: A Gradient descent implementation

7.1.1 Implementation

I have implemented the basic gradient descent approach to estimate the true rank $f = 10$ svd approximation of the rating matrix given by the NetFlix Prize. This showed that the method is very computationally intensive, however after optimization the method runs in time linear in the number of components. Even then it still takes a couple of hours just to run through 200 epochs (training passes) for each of 10 features.

The predictions made by the factorization are conservative, they tend to be closer to the global mean than they should.

The result as messured by rmse is comaprable to the Cinematch system implemented by Netflix eaven without regularization.

The algorithm works as follows:

```
#Features is the rank of the approximation
#Users is the number of users
#Movies is the number of movies
ratings is a collection of Ratings on the form [rating, user, movie]
convergence is a condition to stop training the current feature,
  it is true if number of epochs > 300 or sum(error) < 500
learningrate is a fixed number, here set to 0.001
svd() {
  real[][] userFeatures = new real[#Features][#Users];
  real[][] movieFeatures = new real[#Features][#Movies];
  Initialize userFeatures and movieFeatures to random matrixes.
  for(f in 1 to #Features) {
    while(not convergence) {
      for(Rating r in ratings) {
        error = r.rating - predict(r.user, r.rating, f)
        userFeature[f][r.user] += learningrate*error*movieFeature[f][r.movie]
        movieFeature[f][r.movie] += learningrate*error*userFeature[f][r.user]
      }
    }
  }
}
```

Where the key components are calculating the error and subtracting its influence on the different features.

The predict function predicts ratings using #features first features and is implemented as follows:

```
predict(user, movie, #features)  {
  pred=0;
  for(i in 1 to #features) {
    pred = pred + userFeature[user][i]*movieFeature[movie][i];
  }
  return pred;
}
```

This is simply a row in the userfeatures multiplied by a column in the transposed moviefeatures.

I obtained a $rmse = 0.8716$ on the probe-set (N=1408395), prediction $mean = 3.656$, $var = 0.759$. This unfortunatly turned out to be before the probe set was subtracted from the training set, when this was done the rmse climbed to 0.955, or about the same as the Cinematch score.

If we examine how the predictions are on different true ratings, we get the following results: In figure 3 I have plotted normal-distributions with the means

True rating	freq	prediction mean	prediction var	prediction rmse
1	73211	2.697	0.580	1.860
2	136082	3.005	0.370	1.174
3	352436	3.362	0.284	0.645
4	462093	3.756	0.250	0.555
5	384573	4.216	0.270	0.940

Table 1: Results per Rating

and variances from table 1. In figure 4 I have plotted the relative frequencies of predictions for each real rating. The strange nearly vertical lines to the right are due to the cutoff, ratings are restricted to be in the legal rating span from 1 to 5. The data are taken from the probe-set.

A plot of the residuals(true rating - predicted rating in the probeset) is shown in figure 5 and shows that this resembles a slightly skewed normal distribution.

When plotting rmse vs. features, we see that when no shrinkage is applied, the prediction strength of the svd increases only up to 10 features, and declines afterwards. The plot can be found in figure 6. When shrinkage is applied Simon Funk among others have shown prediction strength to increase well above 50 features.

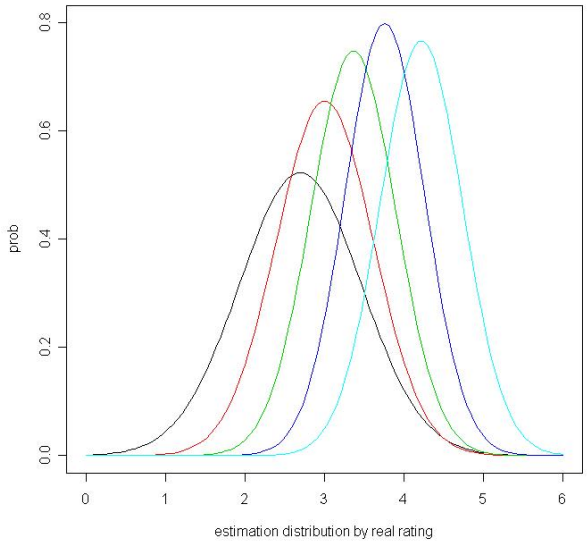

Figure 3: Prediction probabilities

Possible improvements include:

- Centering the ratings by subtracting the pr.movie mean.
- Introducing a non-linear learning function.
- Combining ratings with a apriori estimation of the rating, e.g. the global mean, to reduce the effect of ratings in sparse movies and/or users.

7.1.2 Interpretation

So what is realy going on in theese attempts to reconstruct the matrix A from U and V? When A is complete the question is down to simple linear algebra, but when we have lots of missing values one can and should ask what $R^*_{ij} = U_{(i)} \times V^t_{(j)}$ actually is. An intuitive way of looking at it is that $U_{(i)}$ somehow identifies the weight that user i emphasizes on different qualities in a movie, and $V^t_{(j)}$ somehow encodes the qualities of movie j. We can run a little experiment to check our intuition. By not centering the data we would expect the most prominent qualities of the movie to occupy the first features of it. With any luck we would be able to identify the meaning of the different features belonging to each movie. A speculative guess to the first few features would be overall quality, affinity to genre (amount of violence, romance, comedy...), nationality (language). So let's give it a go:

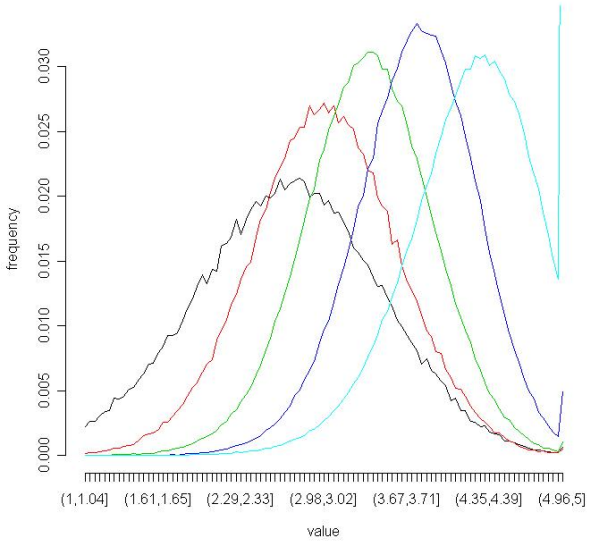

Figure 4: Prediction frequencies for different ratings

I have implemented the Gradient descent SVD, presented in section 7.1, and extracted the experimental data from the results.

A plot of the first movie feature vs. the average rating for a few randomly chosen movies is very convincing, see plot 7. The first feature obviously encodes the average rating. The top and bottom movies ranked by this feature can be found in the appendix, 83. Note that the sign is arbitrarily, so in this specific run it ended with negative scores for the highest ranked movies.

A strategy for understanding the different features could be to look at movies with the highest and lowest scores in the feature to see what is being discrimated. I have listed the 30 top and 30 bottom movies by each feature and try to understand what they encode. The listings can be found in appendix A.

- The top ranked movies occupy the lower end of feature 1, see figure 24. Somewhat suprisingly this is dominated by series, not feature films. Among the lowest rated films, most are totally unknown to mee, except for Dune witch I personally found suprising to be this badly regarded. The Dune movie in question turned out to be a specific collectors edition, only rated 13 times, thereby being a little under determined.

- Movies with highest and lowest scores in feature 2 can be found in figure A. Is there a pattern here? Serious and quality movies dominate the top(except Napoleon Dynamite?), more action oriented, faster moving

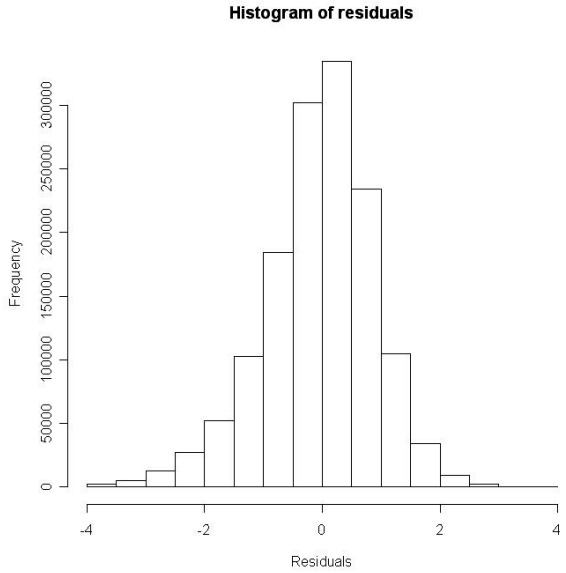

Figure 5: Residuals in a simple 20-feature SVD

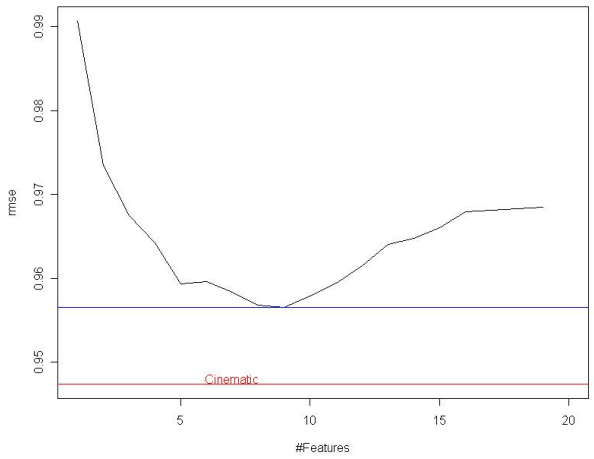

Figure 6: Predictionstrength by number of features.

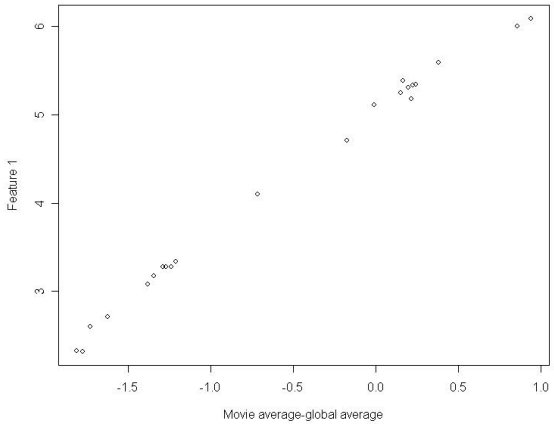

Figure 7: Average rating vs. Feature 1

movies dominate the bottom. Keep in mind that this table is independant of the average rating of a movie as this is accounted for in the orthogonal first feature, assuming that the first feature actually ecodes average rating and the decomposition is actually a true SVD.

- In feature 3, figure 26, we see that "feelgood" series dominate the bottom, and more unpleasant movies occupy the top of the scale.
- In feature 4, figure 27, we see that gay and sexually oriented movies occupy the top, while more hetrosexual (macho) movies dominate the bottom.
- In feature 5, figure 28, it is difficult to find a real pattern, but the difference between the top and bottom is very clear.
- In feature 6, figure 29, The Twilight Zone dominates the upper end together with Dragon Ball, a action anime of the Japanese manga type. Mystery oriented modern movies dominate the lower end.
- In feature 7, figure 30, modern feelgood films dominate the top, and crime and drama from the seventies dominate the bottom.
- In feature 8, figure 31, teenage stuff dominate the top, while sci-fi series dominate the bottom.
- In feature 9, figure 32, Dragon Ball again dominates the top, this time more modern than before, while slightly dark movies dominate the bottom.
- In feature 10, figure 33, horror movies obviously occupy the lower end of the scale.

Although the affinity to the traditional generes is sometimes difficult to spot, we do recognize a lot of the qualities that come into play when separating movies in the different features.

7.1.3 Java implementation

The Singular Value Decomposition is represented by a class, SVD. The features themself are two dimensional floatingpoint arrays, the movie features have $rank \times N$ dimensions, where N is the number of movies, the user features have $rank \times M$ dimensions, where M is the number of users. Prediction is first done by the method predict, seen below.

```
public float predict(int movieIndex, int userIndex, int rank) {
    float result = estimateBaseline(movieIndex, userIndex);
    for(int i=0; i < rank; i++) {
        result+= userFeatures[i][userIndex]*movieFeatures[i][movieIndex];
    }
    return result;
}
```

The *esimateBaseline(movieIndex, userIndex)* calculates the baseline of the prediction, e.g. the sum of the movie mean and the user offset. The resulting value is then clipped to the legal value.

The actual training is done in a seperate class, the training itself is done in the method train, as can be seen below. The method is optimized by caching the last results and errors, this makes the training time linear in the number of features rather than quadratic.

```
private static void train(SVD svd) {
    double lastError = 0;
    float[][] userFeatures = svd.getUserFeatures();
    float[][] movieFeatures = svd.getMovieFeatures();

    int mid[] = ratings.getMovieIds();
    int uid[] = ratings.getUserIds();
    byte[] ratingArray = ratings.getRatings();
    lastPrediction = new float[ratings.getNofRatings()];

    int nofRatings = ratings.getNofRatings();
    float estimate = 0;
    float error = 0;
    double errorInLastFeature = Double.MAX_VALUE;

    //initialize lastPrediction to baseline
    for(int r=0; r < nofRatings; r++) {
        lastPrediction[r]=svd.estimateBaseline(mid[r],uid[r]);
    }
    for(int feature=0; feature < svd.getNofFeatures(); feature++) {
        //Check for stopping condition.
        if(...) {
            break;
        }
```

...continued on next page...

```
...continued
        int epochs = 0;
        for(int i=0;i < svd.getMaxEpochs(); i++) {
            double totalError =0d;
            float learningRate = i==0?svd.getLearningRateFirst():svd.getLearningRate();
            for(int r=0; r < nofRatings; r++) {
                estimate=lastPrediction[r] + userFeatures[feature][uid[r]]*movieFeatures[feature][mid[r
                error = ratingArray[r]-estimate;
                totalError +=Math.abs(error);
                error = error*learningRate;
                userFeatures[feature][uid[r]] += error*movieFeatures[feature][mid[r]];
                movieFeatures[feature][mid[r]] += error*userFeatures[feature][uid[r]];
            }
            lastError=totalError;
            epochs = i;
            if(i > svd.getMinEpochs() && improvement < svd.getImprovmentLimit()) break;
        }
        //Update lastPrediction with the current prediction.
        for(int r=0; r < nofRatings; r++) {
            lastPrediction[r]=lastPrediction[r]
                + userFeatures[feature][uid[r]]*movieFeatures[feature][mid[r]];
        }
        errorInLastFeature = lastError;
    }
}
```

7.2 Logarithmic model

7.2.1 Notation

We introduce the following notation:

- Users are denoted i,j
- Movies are denoted k,l
- Ratings, z, are discrete, $z \in Z$, eg. $Z = \{1, 2, 3, 4, 5\}$
- Ratings of user i on movie k are denoted x_{ik}.
- The set of movies rated by user i is denoted by M_i
- The set of movies rated by both user i and user j is denoted $S_{ij} = M_i \cap M_j$.
- The distance between two ratings is denoted $D(x, z)$.
- The distance between two users is denoted d_{ij}.
- $\beta(\theta)$ is a regression function.

7.2.2 Model

We assume the following as a model for the ratings of a user i on a movie k given user j who has seen k and her ratings:

$$p^k_{i|j}(x) = P(X_{ik} = x | \{(x_{il}, x_{jl}); l \in S_{ij}\}, x_{jk}) = \frac{\exp(\beta(D(x, x_{jk}), d_{ij}))}{\sum_{z \in Z} \exp(\beta(D(z, x_{jk}), d_{ij}))} \tag{11}$$

So, the probability of user i giving the rating x to movie k given information on user j is proportional to an exponential function of the distance between x and user j's rating of movie k and the similarity of the two users.

Overall this gives the following probability of user i giving the rating x to movie k.

$$p^k_i(x) = P(X_{ik} = x | \{\{(x_{il}, x_{jl}); l \in S_{ij}\}, x_{jk}; j \neq i\}) = \frac{1}{n-1} \sum_{j \neq i} p^k_{i|j}(x) \tag{12}$$

7.2.3 Distance functions

Several choices are available for the distances lets list a few with pros and cons.

7.2.4 Box-Cox distance

Let the distance between ratings be the standard Box-Cox distance:

$$D^{BC}(x,y) \stackrel{def}{=} \frac{|x-y|^{\alpha}-1}{\alpha} \tag{13}$$

This has the advantage of making the transformed distances more symmertric, thereby fulfilling the normal requirements of most regression forms better. It also allows us to move the weight of the rating distances to fit out model. It does not take into account that different users have different ways of applying ratings.

The standard form does not fit well with our data since it gives a negative distance when the ratings are equal, we remedy this by modifying its form, to:

$$D^{BC}(x,y) \stackrel{def}{=} \frac{|x-y|^{\alpha}}{\alpha} \tag{14}$$

We can define the distance between users as average Box-Cox distances:

$$d_{ij}^{BC} \stackrel{def}{=} \frac{1}{||S_{ij}||} \sum_{l \in S_{ij}} \frac{|x_{il}-x_{jl}|^{\alpha}}{\alpha} \tag{15}$$

7.2.5 Free parameter distance

We can also let the distances between ratings be free parameters in the model, eg let $\alpha_0, ..., \alpha_4$ be real numbers restricted to $\alpha_0 < \alpha_1 < ... < \alpha_4$ and let the distance between two ratings x, y be $D(x,y) = \alpha_{|x-y|}$

The distance between users can still be the average rating distance between them, $d_{ij} = \frac{1}{||S_{ij}||} \sum_{k \in S_{ij}} \alpha_{|x_{ik}-x_{jk}|}$.

7.2.6 Regression form

Finally we must come up with at suitable regression function, it should reflect the relationship between a hyphothetical rating, x, and someone elses rating together with the distance to this other user. We want β to be large when x is close to x_{jk} and the distance to user j is small. Perhaps something like this:

$$\beta(D(x,y), d_{ij}; \theta) = \beta_0 + \beta_D D(x,y) + \beta_d d_{ij} + \beta_{Dd} D(x,y) d_{ij}$$

Where θ consists of $\beta_0, \beta_D, \beta_d, \beta_{Dd}, \alpha_0, ..., \alpha_4$. We still have not accounted for the relative support of each user j, the number of movies that both i and j has seen.

7.2.7 Is this a Generalized Linear Model?

We start by comparing our current model with the Exponential family of distributions. This will place some restrictions our regressionform. A probability distribution is a member of the Exponential family of distributions if it can be written in the following form:

$$f(y,\theta) = \exp(a(y)b(\theta) + c(\theta) + d(y))$$

or

$$\log f(y,\theta) = a(y)b(\theta) + c(\theta) + d(y)$$

First we show that our conditional distribution (11) is of the Exponential family.

$$\log(p^k_{i|j}(x)) = \beta(D(x,x_{jk}),d_{ij}) - \log\sum_{z\in Z}\exp(\beta(D(z,x_{jk}),d_{ij}))$$

The second term is not a function of x, so this is part of our $c(\beta)$ function. The first term written out with our suggested regresionform comes to:

$$\begin{aligned}\beta(D(x,x_{jk}),d_{ij}) &= \beta_0 + \beta_D D(x,x_{jk}) + \beta_d d_{ij} + \beta_{Dd} D(x,x_{jk}) d_{ij} &(16)\\ &= D(x,x_{jk})(\beta_D + \beta_{Dd} d_{ij}) + \beta_0 + \beta_d d_{ij} &(17)\end{aligned}$$

Again the bits after the first term are not dependent on x, so we include them in our $c(\beta)$. The first term is on the form $a(x)b(\beta)$, where $a(x) = D(x,x_{jk})$ and $b(\beta) = \beta_0 + \beta_{Dd}d_{ij}$. We conclude that (11) is on the Exponential form. However, to be a GLM, we also need it to be canonical, $a(x) = x$, this is not compatible with a sencible distancefunction. We conclude that our conditianal probabilities is on exponential form, but not a GLM.

To check wheter our unconditional probabilities are on exponential form we look at the expression:

$$\log(p^k_i(x)) = -\log(n-1) + \log\sum_{j\neq i} p^k_{i|j}(x)$$

This is unfortunatly not on exponential form, but the good news is that it is still concave.

7.2.8 Accounting for common support

At this point we have a regression form that weights each movie seen by another user proportional to some function of the distance to this user. We do not consider the number of movies seen by both the user in question and the owner of a certain rating. One would think that we should put more weight into users

that are looking similar and has a large support for that assumption, that is d_{ij} is small and $||S_{ij}||$ is large, perhaps something like

$$p_i^k(x) = P(X_{ik} = x|\{\{(x_{il}, x_{jl}); l \in S_{ij}\}, x_{jk}; j \neq i\}) = \frac{1}{n-1} \sum_{j \neq i} w_{j|i}\ p_{i|j}^k(x)$$

where $w_{j|i} = \frac{c+||S_{ij}||}{\sum_j c+||S_{ij}||}$ and c is either a constant eg. 1, or a free variable.

7.2.9 Maximum Likelihood

We would like to find the optimal values of θ for the regression function. This can be accomplished by Maximum Lilekihood Estimation (MLE) as shown in the following.

We define the Likelihood function as:

$$L(\theta) = \prod_{i=1}^{n} \prod_{k \in M_i} p_i^k(x_{ik})$$

Our task is then to find the $\hat{\theta}$ that maximizes this function,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta),$$

and since argmax is insensitive to monotone transformations we go a bit further with

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \log(L(\theta)) = \underset{\theta}{\operatorname{argmax}}\, l(\theta) = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{k \in M_i} \log(p_i^k(x_{ik}))$$

Since our model is not a GLM, and not strictly on the exponential form we have to do a bit of work to find the MLE. Luckily this is a concave function as can be seen from observing that the second derivates with respect to β is negative definite. Although concave, the function is not smooth enough to apply gradient ascent to find its maximum. Instead we take advantage of the Non-Linear Minimization (nlm) method in R whitch uses a Newton-type algorithm to minimize the supplied function. Unfortunatly the capabilities of R does not extend to handling millions of ratings, so we are limited to test the method on a smaller dataset. The dataset used in the following comes from eSmak as introduced in section 3.4.1, and contains all users with more than 15 ratings and all movies seen by these users. The set contains 318 users and 723 movies, with at total of 11473 ratings. Even this modest amount of data takes a long time to handle in the normally fast R-language. The reason being the large amount of interdependencies present in the model. As an example, we take closer look at the unconditional probability of a user i giving a rating x to a

movie k, $p_i^k(x)$, given in equation 12. We see that this depends on all other users, and their distance to i, and their distance to i is based on all movies they have in common, and all possible ratings. The selection of subsets makes matrix notation difficult to use in the different calculations and thereby slows down the algorithm considerably. My present R-script uses more than 48 hours to handle the dataset above, while the presented SVD-CF in section 7.1 takes about 7 hours on a dataset that is almost 9000 times as large.

The R-script can be found in appendix B.

7.3 Metric Neighbourhood Predictor

7.3.1 Idea

A while ago I read a short blog post titled "Evolution and Wisdom of Crowds" [6], this more or less serious post suggested an implementation for the Netflix-Prize, this has been expanded upon and the result is presented in this section.

The basic idea is to place movies and users in a common highdimensional space, and attempt to arrange them in such a way that users who like a specific movie is located near that movie, and users who dislikes a certain movie is positioned some distance away from that movie. We hope this arrangement will also put the users who would like a yet unrated movie close to it. And as a secondary result we hope that users end up in a neighbourhood of similar users, and movies end up in a neighbourhood of similar movies.

7.3.2 An implementation

When trying to implement a Metric Neighbourhood Predictor, we attempt to arrange each user and item into a metrical space in such a way that the opinion the user has of an item is reflected in the distance between them. So, if a user likes a certain item, he/she/it should be positioned close to that item, if the item is disliked by the user there should be some distance between them. In the following we specifically look at users and movies with their associated ratings.

The idea is to create a suitable dimensional space and insert a point for each movie and each user. We then view the known ratings as constraints, specifically a rating-tripple (u, m, r) by user u on movie m giving the rating r is viewed as constraining the distance between the user u and the movie m to $f(r)$ where f is some positive function.

In this example we use

- An $N \in [1, 350]$ dimensional Euclidian space.
- $f(r) = K - r$, where $K > \max(r)$ meaning that films a person like should be close to that person, and disliked films should be further away.
- We initialize the movie-positions and user-positions to some random location, eg. users are in $-5\times[u_1, u_2, ..., u_N]$ and movies are in $5\times[u_1, u_2, ..., u_N]$, where the u_i is uniform on $[0, 1]$.

Now we seek to arrange the movies and users in such a way that $\sum |d(u_i, m_i) - f(r_i)|$ is minimized. Our strategy to accomplish this is simply to iterate over the known ratings, and adjust the positions of the user and the movie to more closely match the actual rating. This is repeated until convergence.

In figure 8 we see an example with only 6 users (red dots) and 10 movies (blue dots) arranged in a two dimensional Euclidian space, the known ratings are pictured as edges between the user and the movie. *Green* edges have length similar to the actual ratings, *red* edges indicate that the user and movie is too close and finally *blue* edges indicate that the distance between the user and movie is grater than f(r) where r is the known rating. The snap-shot is taken after 30 iterations. In figure 9 an equilibrium has been reached and all constraints are satisfied, this took about 100 iterations in this example, but this number varies greatly.

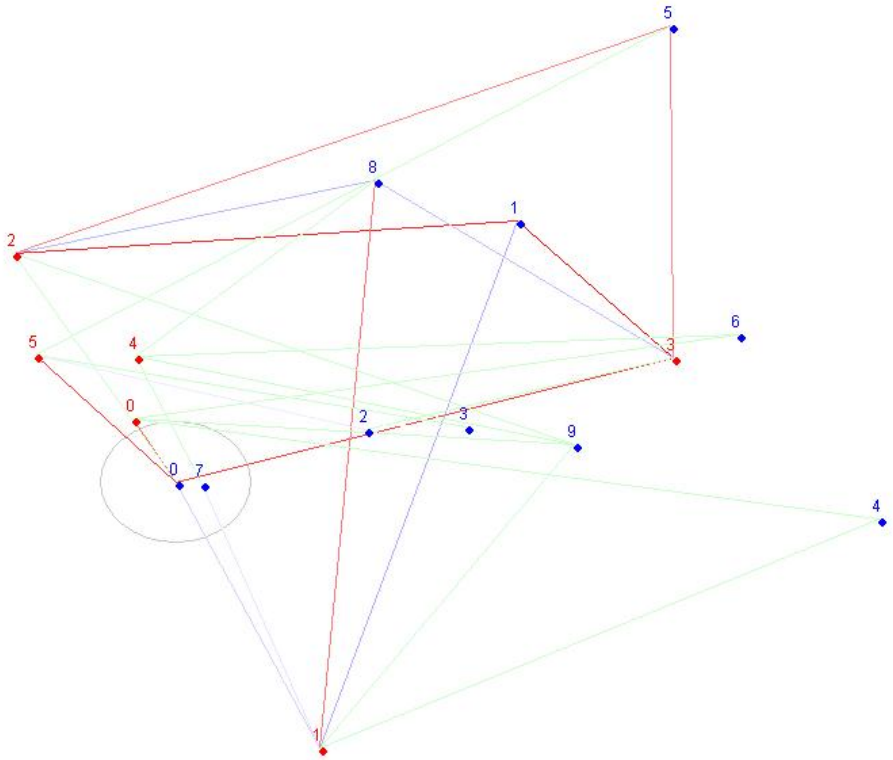

Figure 8: Metric Neighbourhood Predictor - Pre equilibrium

We use the constellation to predict unknown ratings by taking $\hat{r_{ij}} = f^{-1}(d(u_i, m_j))$, so with our choise of f, the closer the movie is to the user, the better the user is predicted to like the movie. In figure 9 user 1(red dot, bottom left) is predicted to like movie 3 (blue dot middle) somewhat, but to dislike movie 5(blue dot, top) strongly.

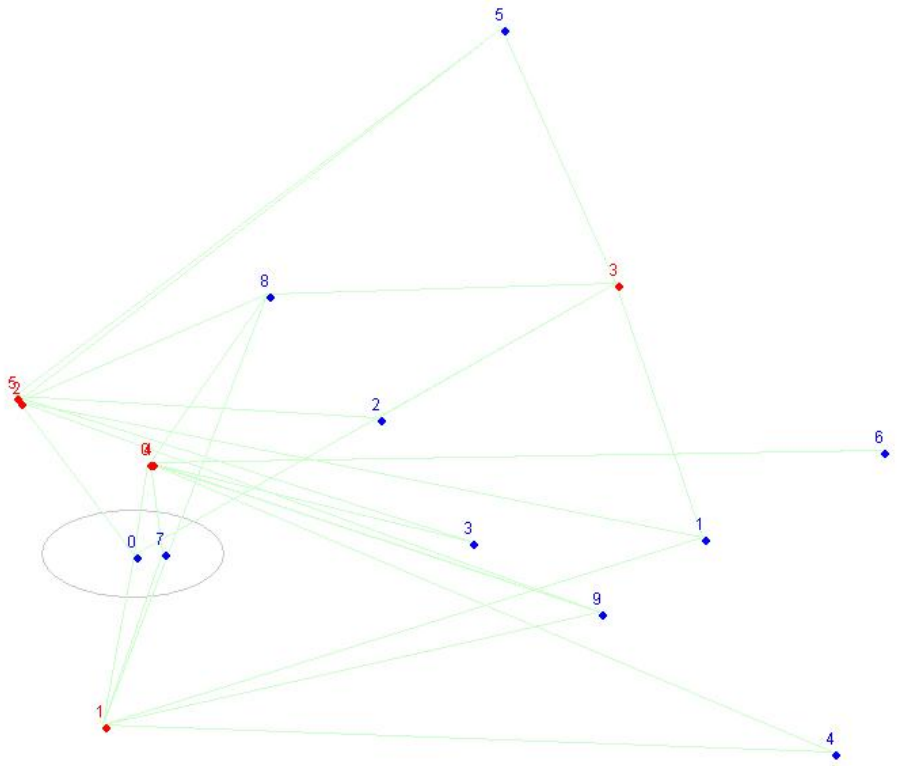

Figure 9: Metric Neighbourhood Predictor - Equilibrium

7.3.3 Algorithm

The training algorithm is straight forward:

```
#l - is a learning rate, eg. 0.1
#g - is the inverse of f
#d - is the distance function
#u - is the userposition vector.
#r - is the rating.
#m - is the movieposition vector.
While not converged
  For each rating-tripple (u,r,m)
    factor <- l*[r-g(d(u,m))]
    u <- u + (m-u)*factor
    m <- m + (u-m)*factor
  end for each
end while
```

There are a number choices to be made about the algorithm.

- How to initialize the positions of users and movies.
- The learning rate.
- The distance function, eg. Euclidian or City Block distance.
- The update order, random or predetermined. Also we could do a simultanous update, acumulating the movements for all ratings and then update the positions for each movie and user.
- The dimensionality.
- The convergence criteria. This is mostly set to a time-span in practice, typically we iterate for about 5-20 hours and stops there.

7.3.4 Problems and Predictions

For a given set of known ratings it is not always possible to satisfy all constraints in a low-dimensional space, this can always be overcome by increasing the dimensionality (but not necessarily with the learning algorithm above). This is assuming there are no inconsistent ratings (eg. a user rates the same movie twice, with different ratings). This can be shown easily by considering the following scenario:

1. Let the number of dimensions equal the number of movies.
2. Put all the movies, $m_1, m_2, ..., m_M$ in origo.
3. For each user, u, let its i'th dimension have the value $f(r_{u,m_i})$ where r_{u,m_i} is r if there exist a ratingtuple (u, m_i, r) and 0 otherwise.

Of course this would lead to a totally uninteresting arrangement. A famous slogan from the machine learning environment is that "compression equals understanding". Following this slogan, we try to reduce the number of dimensions drastically.

Although it is obvious that there exists a, possibly not unique, global minimum, the algorithm does not always converge towards this. There are a lot of local minimas to get stuck in, typically situations like in figure 10 where both user 1 and 2 should be closer to movie 0 (in origo) but is held back by their constraints to movies 1 and 8. When the local minima are not too deep the algorithm is able to escape, this is illustrated in the rmse-plot, see 11. Here you can see that two distinct local-minimas has been avoided.

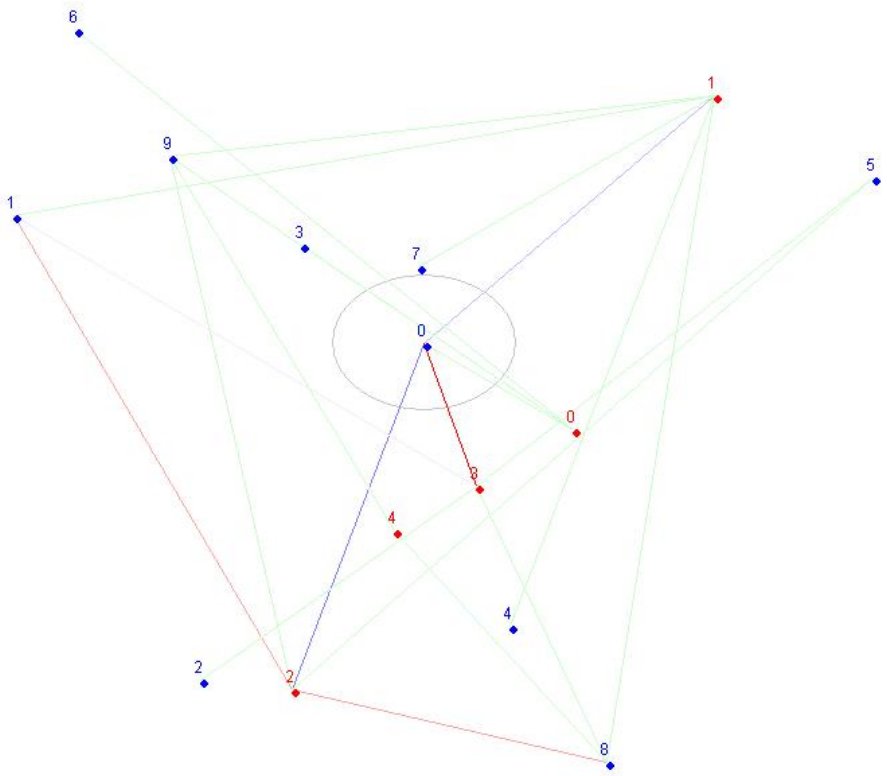

Figure 10: Metric Neighbourhood Predictor - Local Minima

So how do we predict the preference of user u for movie m, that is not in the ratingset? The general idea is to just take the inverse of the f-function above with the observed distance, $d(u,m)$, as its parameter. $\hat{r_{u,m}} = f^{-1}(d(u,m))$. Ideally we want situations like in figure 9, where user 5 and 2 (the two red dots almost on top of each other, to the far left) are very similar users, they both have rated movies 5 and 0 equally. However user two also has rated movie 8, and in this equlibrium state the distance to this movie determines user 2's rating of movie 8. So a good estimate og user 5's rating of movie 8 is intuitively given by the users distance to movie 8. But again there are problems, look at movie 6 in the same figure, its positions is restricted by only one constraint, a rating by user 0, and this constraint is satisfied on the full circle centered on user 0. So in this layout user 2 rather dislikes movie 6, but that is completly arbitrarily, since movie 6 also could be very close to user 2 while still satisfying its only constraint. In general, every point with fewer constraints than there are dimensions in the space can be placed anywhere on a hypersphere in N - #constraints + 1 dimensions centered on the sum of constraints, see fig 12. So

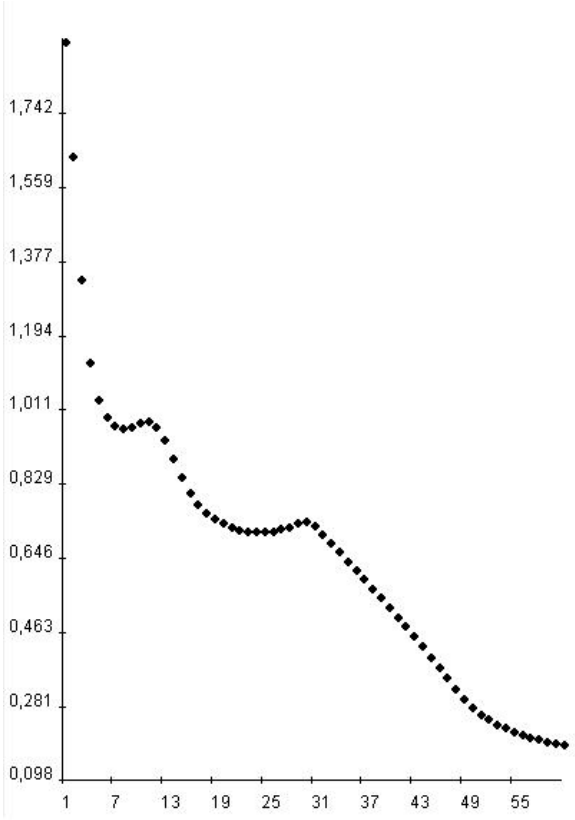

Figure 11: Metric Neighbourhood Predictor - RMSE by iteration. Two local minimas

a point determined by a number of constraints equal to the dimensionality of the space could be located in two dinstinct points, eg. user 2 (red, bottom) in figure 12 could occupy the position it holds in the figure, but would also satisfy its constraints in the position of movie 1.

We call this phenomenon *underdetermined positions*.

Another problem occure when the number of constraints is much larger than the dimensionality, they can in general not all be satisfied, and the effect for a particular item is not very predictable. A look at the mean root squared error of predictions on a rating set distinct from the training set in a 300 dimensional space graphed vs. user support, the number of movies each user has seen tells an interesting tale. First presented is a plot restricted to ratings made by users with less than 500 ratings, this is shown in figure 13. This looks as we would hope, the more movies the user has seen, the better we can predict ratings, and hence the error falls with the support. But when we see the full picture, including

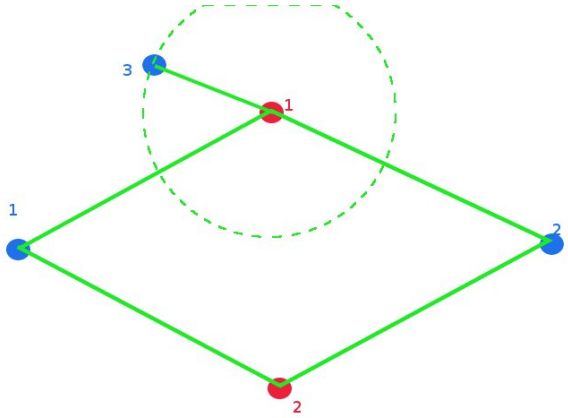

Figure 12: Movie 3(blue) is underdetermined

users that have rated more than 500 movies the problem with overdetrmined positions shows it face, look at figure 15. Here we see that the prediction errors varies greatly and apparently randomly. The quantities involved is part of the equation, the groups with large support is generally smaller than the ones with small support, therby their mean varies more. There are a few distinct outliers in figure 15, all supported by 4000 movies or more, without investigating the matter further, my guess is that there are some users that have entered a lot of more or less random data, and the number of users in each of these groups is small enough to make them outliers. This is supportet somewhat that the groups of users with less than 500 movies support are very well behaving. Finally we note that the points to the far right are all single users.

For completeness a histogram of usersupport is presented in figure 14, where the 3719 users who have seen more than 2000 movies are capped to 2000.

7.3.5 Metric Neighbourhood Predictor - Mixture

A natural soulution to the problem of underdetermined positions is to train a lot of nets and use some kind of mean on these to predict new user-movie ratings. This would eliminate the arbritrariness of their position, and falls well under the idea of "The Wisdom of Crowds".

7.3.6 Results

I find that the prediction quality increases with the number of dimensions, at least upto 50, see figure **??**. After that, training time becomes an issue, more epochs are needed, and each takes longer to compute when the number

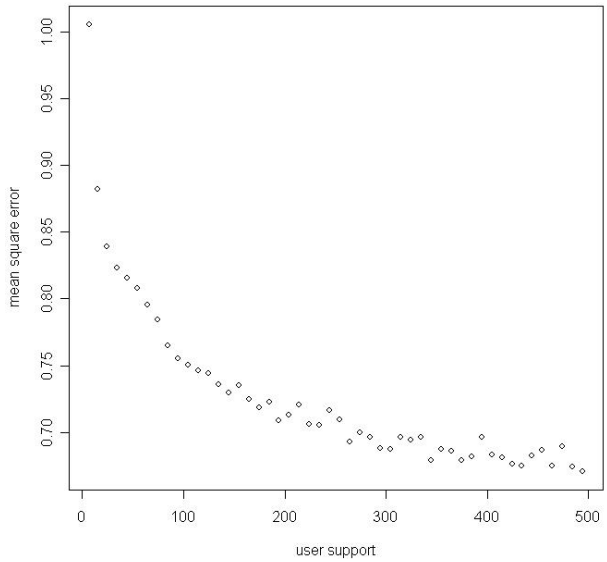

Figure 13: Error by user support, users with less than 500 ratings

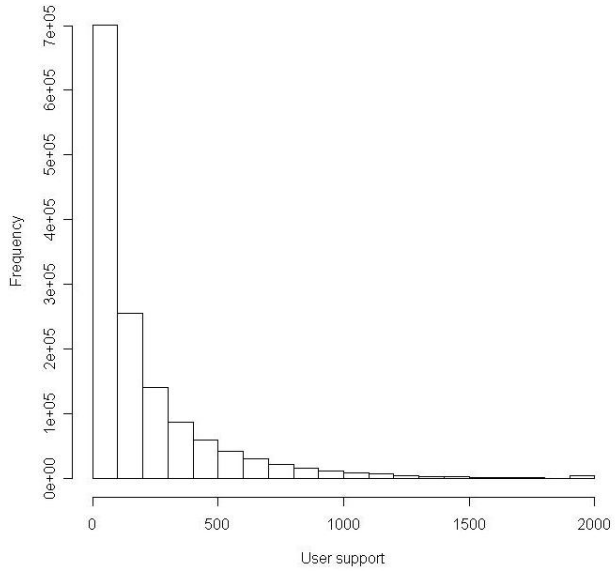

Figure 14: Histogram of usersupport, capped to 2000

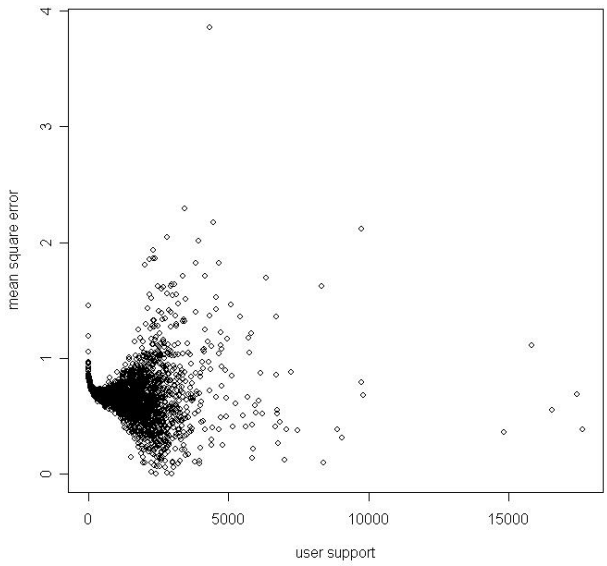

Figure 15: Error by user support, unrestricted

of dimensions increase. In a net of 300 dimensions, whitch is pretty much the limit for my machine resources, the convergence is quite fast as can be seen in figure 17. After about 20 iterations we are close to the achieved minimum. In figure 17 the full netflix-dataset is used, and as can be seen there are no signs of the local minima apparent in figure 11, this is assumed to be caused by the share number of local minimas, and by the fact that the local minimas is local also in the sense of only affecting a neighbourhood of users and movies.

With a dimensionality of 300, witch is the one plotted in figure 17, we obtain a training set rmse of 0.8312, and a probe set rmse of 0.95377 whitch is slightly worse than the Netflix Cinematch rmse of 0.9474. Due to time restraints I have only trained the net in 91 epochs (this took 9 hours, or 6 minutes pr. epoch), and due to memory restraints I had to limit the dimensionality to 300(requiering about 150 million floating point numbers to position movies and users).

The rmse plotted in figure 17 is not the true trainingset rmse, rather it is calculated after adjusting the positions of users and movies after each rating, therby looking a bit better than the true trainingset rmse.

It is interresting to see how movies position themself relative to one another. In figure 18 I have plotted the positions of some selected movies projected down on two arbitrarily chosen dimensions. We can see that the romantic movies, encircled in pink, are lumped together in the horizontal dimension, but are covering the full length of the vertical dimension. The three thrillers(A Clockwork Orange, 12 Monkeys, 9 1/2 Weeks) are gathered at the top. The three epic

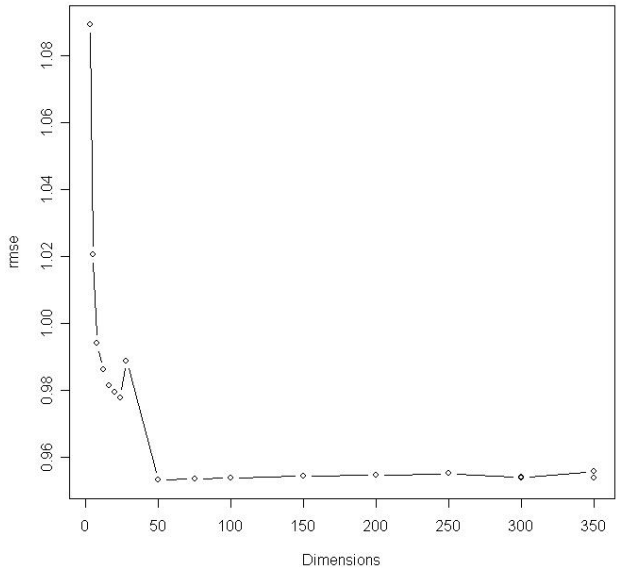

Figure 16: Rmse by dimension in MNP

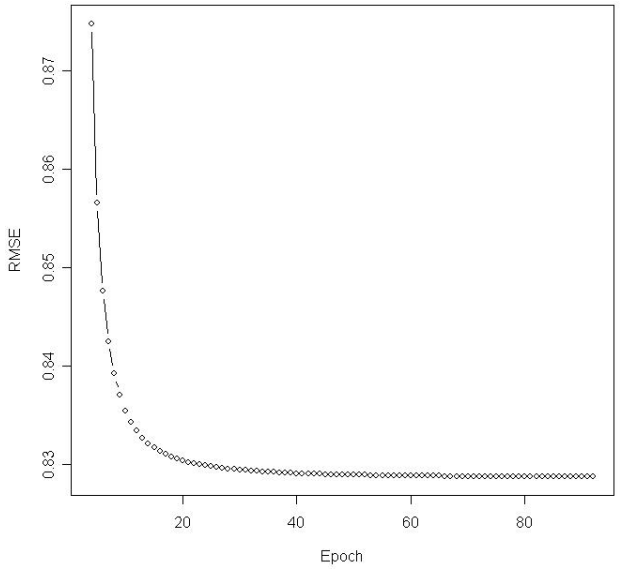

Figure 17: Training an onp: rmse by epoch(from epoch 2)

Chinese movies(The Last Emperor, Hero, Crouching Tiger Hidden Dragon) are somewhat entangled with the thrillers(12 Monkeys, A Clockwork Orange, 9 1/2 Weeks), but are grouped nicely together. The two brown ellipsis enclose four of the Star Wars movies, the ones in the bottom ellipsis are the new generation, while the top brown ellipsis enclose the old generation. Also in the viscinity of the top brown ellipsis we find all the Lord Of The Rings movies.

When solely plotting the distance to each of the same movies from a selected one, in figure 19, I selected The Lord of the Rings: Fellowship of the Ring, one sees the ability to position at least very similar movies close together.

7.3.7 KNN in Metric Neighbourhood Predictor

When a metric space is available, a natural method to apply is the K-Nearest Neighbours. This is relativly easy to implement in the framework of a MNP as the distance function is a natural part of our model. I implemented a weighted KNN, where the weights are a logit like function of the distance. So to predict a single rating $\hat{r}_{im}$, for user i and movie m the algorithm works as follows.

1. Create the set $M(m)$ of all users who have seen movie m.
2. Calculate the distance from user i to each of the users in $M(m)$.
3. Create the set $M_i^K(m)$, consisting of the K users in $M(m)$ closest to i
4. The prediction is then

$$\hat{r}_{i,m} \frac{\sum_{j \in M_i^K(m)} r_{jm} w_{ij}}{\sum_{j \in M_i^K(m)} w_{ij}}$$

The actual form of the weights used was $w_{ij} = 1 - \frac{1}{1+\exp(-d(i,j)/2)}$. As can be seen, this method does not need any aditional training when an MNP is available. The time to make a prediction is quite high, in absolute terms it takes about half a second on my computer, meaning that a pass through the probeset of 1.5 million ratings takes more than 10 days. The limited amount of testing I have done on this gives reason think it is not a very successfull approach, the rmse reported on the first few thousand ratings point to about rmse=1 whitch is very bad. When running the algorithm, it is very clear that it fits some movies/users better than others. So I tested to see what happened when only predicting ratings where the user had a close neighbourhood. I calculated an rmse on only the cases where the user had relevant neightbours, this was implemented as $\sum_K w_{ij} > \sum_K 1 - \frac{1}{1+\exp(-0.95/2)}$, that is the average weight should be less than w^*, the weight of a user 0.95 away from our subject. The

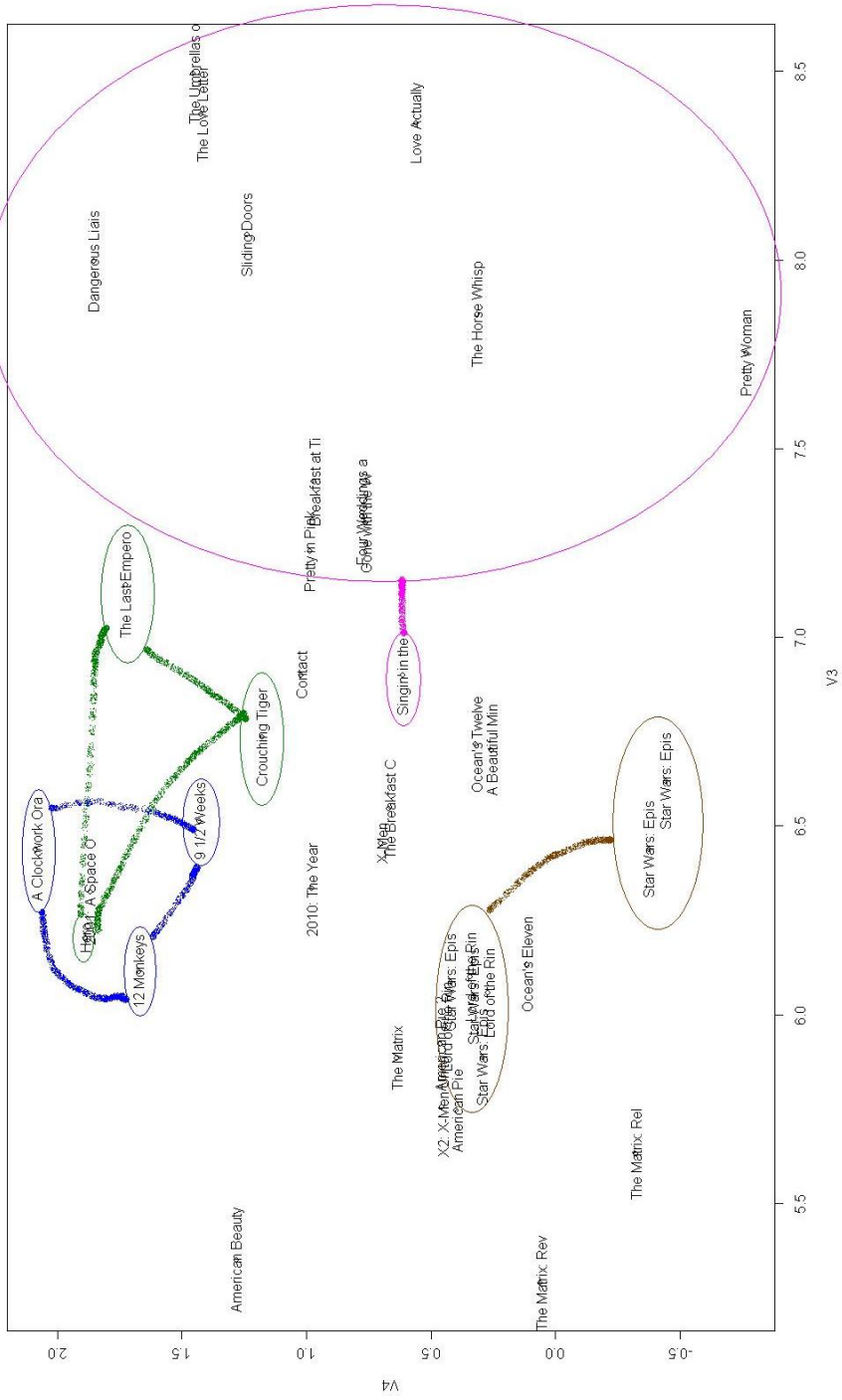

Figure 18: Positions of selected movies in an MNP

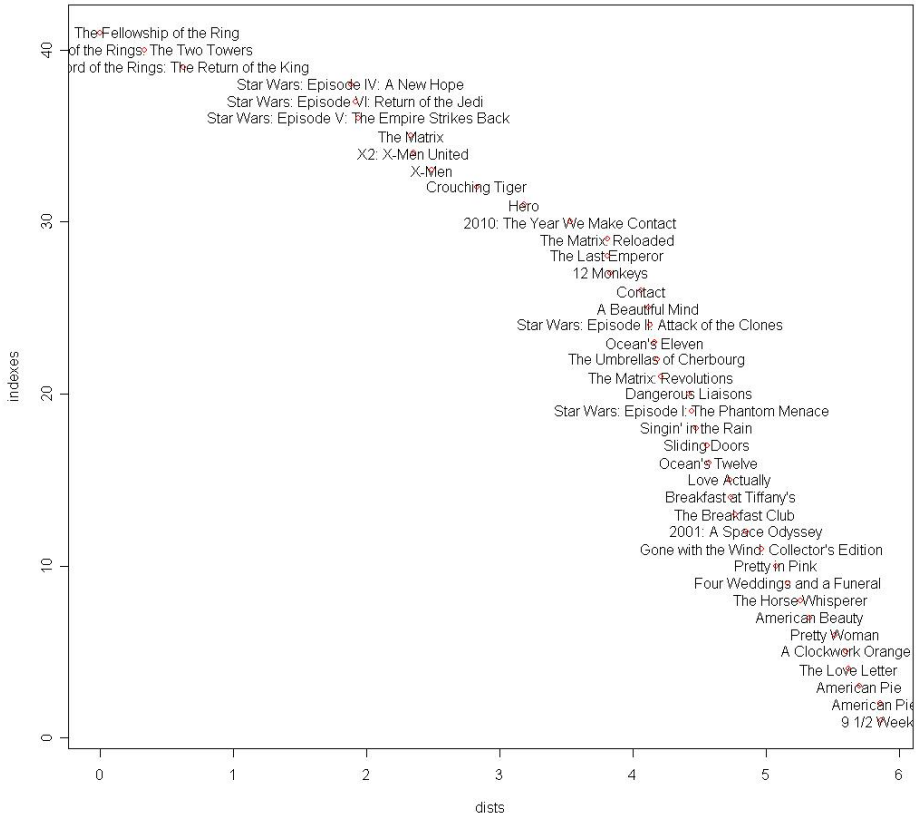

Figure 19: Distance from Lord Of The Rings in MNP

number 0.95 is chosen from studying some histograms of user-user distances, e.g. figure 20 and 21. The prediction frequency was only about slightly above 1 in 10, but the rmse on this selected set was impressive:

Dim	**Freq**	**#Pred**	**rmse - selected**	**rmse - all cases**
50	0.23	1200	0.883	0.953
300	0.11	650	0.895	0.954

(the table collums are the dimensions of the model, what proportion satisfices the criteria for prediction, number of predictions made, rmse of predictions made, the total probe set rmse.)

The hightened frequency of predictions even when keeping the same conditions (mean distance < 0.95) is due to distances expanding with the logarithm of the dimensions. It is not given that this selection of ratings are the same as other MNP's, but this is likely. Other predictors might have performance indepentent of this selection, thus making this a valuable contributor to an ensamble of predictors.

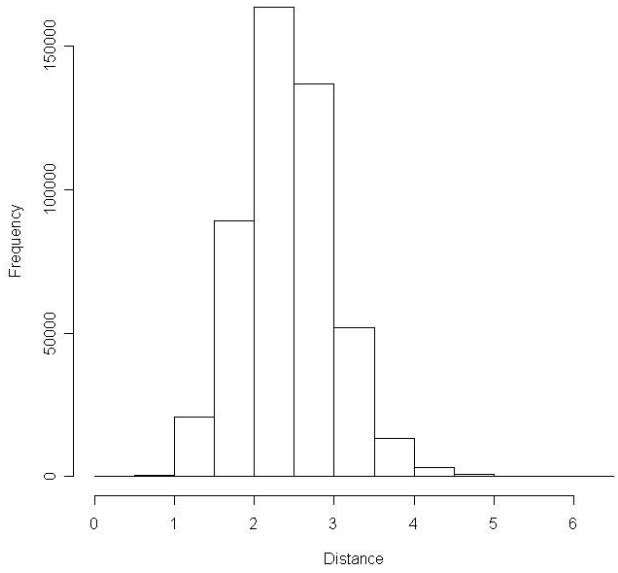

Figure 20: Histogram of distances to other users for arbritrary user

7.3.8 Java implementation

The model is represented by a class *OrganicNeighbourhoodPredictor*, an early name on the method. The positions of the movies is represented as a two dimensional array of size $M \times dim$, where M is the number of movies, and dim is the number of dimensions in the model. The positions of the users are represented

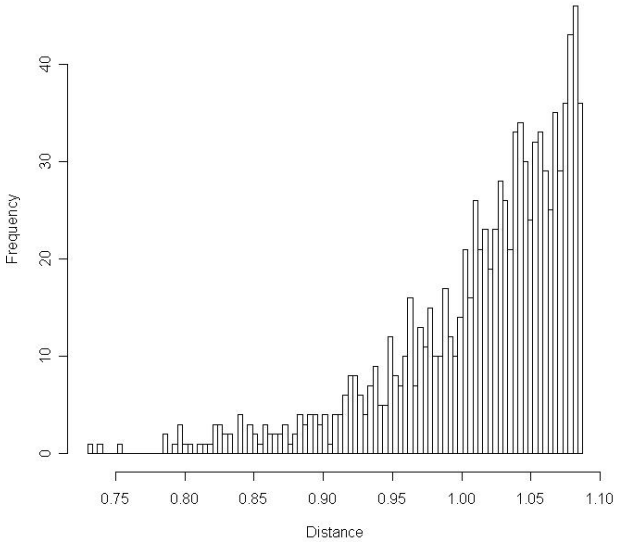

Figure 21: Distances to the 1000 closest users

in an analogous way. The class further contains a *metric*, capable of calculating the distance between two arbritrary points, and the function $f(d)$, enforcing the topology of the net. The training is done in simple passes, adjusting each pair of movies and users encountered in a rating according to closer match $f(d)$. The method is given below:

```
public double train(int[] movieIndexes, byte[] rating,
            int[] userIndexes, int[] order) {
    float targetDist =0;
    double distance = 0;
    double se = 0;
    int index=0;
    double fraction = 0;
    float minRatingDistance = getMinRatingDistance();
    for(int r=0; r < rating.length; r++) {
        index = order!=null?order[r]:r;
        targetDist = Rating.MAX-rating[index]+minRatingDistance;
        distance = metric.dist(
                moviePositions[movieIndexes[index]],
                userPositions[userIndexes[index]]);
        fraction = Math.min(maxFraction,
                Math.max(-maxFraction, delta*(targetDist-distance)));
        move(userPositions[userIndexes[index]],
                moviePositions[movieIndexes[index]], fraction);
        if(movieIndexes[index] > 0) {
            move(moviePositions[movieIndexes[index]],
                userPositions[userIndexes[index]], fraction);
        }
        double temp = targetDist - distance;
        se+=temp*temp;
        }
    return Math.sqrt(se/rating.length);
}
```

The condition

```
if(movieIndexes[index] > 0)
```

is to ensure the net stays anchored in origo. This is acomplished by never moving the very first movie.

As can be understood by the model, there are no way of cashing the results for parts of the calculations, so linear time in the number of dimensions is expected in training of each pass. Also the number of passes needed increases with the number of dimensions. The KNN in Metric Neighbourhood Predictor implementation is done by a seperate class, *KNNByONP*. The prediction method uses an OrganicNeighbourhoodPredictor as its metric.

7.4 Quantile inference

Some example histograms of user ratings can be seen in figure 22. From looking

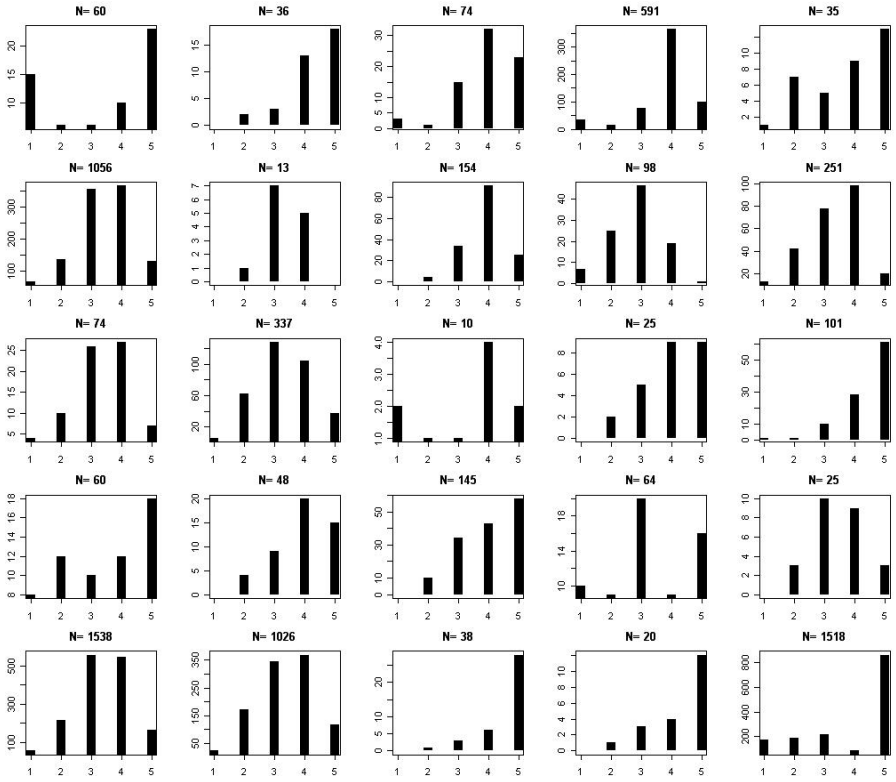

Figure 22: Rating histograms for random users

at lots of histograms of user ratings I belive different users put very different meaning to the ratings. Some prototypes are:

- A *BIASED* user takes the film for what it is, but since he has chosen to see it, he probably likes it. The body of the ratings will be around 4, with a small tail towards the lower ratings (Histograms 2,3 and 5 in the first row).

- An *UNBIASED* user sees a film as a sample from the movies he has chosen to see, meaning that he should get something like a normal around 3 (Histograms 2 and 3 in the last row).

- An *EDUCATOR* user tries to educate the algorithm, gives only 1's, "dont like", 3's, "indifferent" and 5's, "like" (Histogram 1 in first row, and 4 in fourth row).

- A *POSITIVE* user only give 4's and 5's (Histogram 4 and 5 in the third row, and 3,4,5 in the last row).
- A *NEGATIVE* user rearly gives more than 3 (No good examples here, but they do exist).

An idea to cope with the different users is to not regard the given ratings as absolutes, but rather see them as quantiles in the empirical rating distribution of the user. If we model over theese quantiles instead of actual ratings we would then interpet a prediction as a quantile in our target users empirical rating distribution, and can compute the actual rating from this.

A rating a of some user u is to be taken as the quantile of a in f_u. And back, when a prediction is made, lets say we for user v gets the prediction p, then the rating of that prediction is $f_v^{-1}(p)$.

7.4.1 Implementation in Metric Neighbourhood Predictor

I have implemented this regime in the Metric Neighbourhood Predictor setting. The distancefunction then becomes $d(u,m) = C - q(u,m)$, where $C \geq 1$ is a constant, and $q(u,m)$ is the quantile of the rating given by user u on movie m in the user's rating distribution. The results however was less than impressive.

7.5 Blending the results

Different methods are suggested for blending the results of different predictors. The most common is linear regression.

The setting is based on n models, $\Gamma_1, ..., \Gamma_n$ predicting the same training set, producing a data set of $\Gamma_1(i,j), ..., \Gamma_n(i,j), r_{ij}$, where r_{ij} are the known ratings. The procedure is then to run a linear regression where the r_{ij} is treated as the response and the $\Gamma_k(i,j)$ are the covariats. A blended predictions is then $\hat{r}_{ij} = \frac{1}{\sum_{i=1}^{n} \alpha_i} \sum_{i=1}^{n} \alpha_i \Gamma_i$, where α_i are the coefficients from the linear regression. Another use of the same coefficients is a voting scheme where each model predicts a weighted vote, the rating with the highest voting weight wins. I have run a simple linear regression scheme on the results of a SVD model and a MNP model. The result was a modest improvement over the SVD predictions that was the best two.

I have also implemented a basic perceptron with one hidden layer based on backpropagation, where the inputs are the predicted values of the different models, and the movie and user support of each prediction. Unfortunately the learning algorithm did not work, and time ran out as I was bug hunting. Incidently I had the idea of expanding on the SVD with non-linear inferences made by the perceptron using the decomposition of each user and movie as input, due to the same problem as above, this was not completed.

Bagging and Boosting needs to be evaluated in this context, but this has not been done.

8 The Pigeonhole Bootstrap

> Bootstrapping is the process of resampling data to estimate statistics.

This section is a resume of Art B. Owens article The Pigeonhole Bootstrap [10], with some expansions and comparisons.

8.1 Notation

The row entities are $i = 1, ..., R$ and the columns are $j = 1, ..., C$. The variable $Z_{ij} \in \{0, 1\}$ takes the value 1 if we have data for the (i,j) combination and 0 otherwise. The value $X_{ij} \in \Re^d$ holds the observed data when Z_{ij} is 1 and otherwise is missing. We will only work with d=1.

$n_{i\bullet}$ is the number of datapoints in row i. $n_{i\bullet} = \sum_{j=1}^{C} Z_{ij}$

$n_{\bullet j}$ is the number of datapoints in column j. $n_{\bullet j} = \sum_{i=1}^{R} Z_{ij}$

$n_{\bullet\bullet} = \sum_{i=1}^{R} \sum_{j=1}^{C} Z_{ij}$ is the total samplesize.

This arrangement of a $C \times R$ matrix can also be represented by a $N \times 3$ matrix, S, with row $\eta = (I_\eta, J_\eta, X_\eta)$ for $\eta \in 1, ..., N$ with X_η is the same as $X_{I_\eta J_\eta}$ from the previous notation.

We define the ratios $\nu_A = \frac{1}{N} \sum_{i=1}^{R} n_{i\bullet}^2$ and $\nu_B = \frac{1}{N} \sum_{j=1}^{C} n_{\bullet j}^2$. The value ν_A is the expectation of $n_{i\bullet}$ when i is sampled with probability proportional to $n_{i\bullet}$. If two, not necessarily distinct, observations with the same i are called row neighbors, then ν_A is the average number of row neighbors for observations in the dataset. Similarly ν_B is the average number of column neighbors.

We also define $\mu_{\bullet j} \equiv \frac{1}{N} \sum_i Z_{ij} n_{i\bullet}$ and $\mu_{i\bullet} \equiv \frac{1}{N} \sum_j Z_{ij} n_{\bullet j}$. Here $\mu_{\bullet j}$ is the probability that a randomly chosen data point has a row neighbour in column j, and analogous for $\mu_{i\bullet}$.

8.2 Random Effect Model

We consider the data to have been generated by a model in wich the pattern of observations has been fixed. *Does this exclude the importance of what data actually exists in the matrix? E.g. ignore the importance of a datapoints existance, E.g. ignore the importance of a user actually seeing a film?.*

For (i,j) where $Z_{ij} = 1$ we assume

$$X_{ij} = \mu + a_i + b_j + \epsilon_{ij}$$

Where μ is an unknown fixed value and a_i and b_j and ϵ_{ij} are random.

Does this mean that every column and row must have the same mean? Arent our models more like $a_i \sim N(\mu_{A(i)}, \sigma^2_{A(i)})$, this will however not affect the variance estimates below I think.

In classical random effects model its supposed that $a_i \sim N(0, \sigma^2_A)$, $b_i \sim N(0, \sigma^2_B)$ and $\epsilon_{ij} \sim N(0, \sigma^2_E)$, all independently. Here we relax the model and assume only $a_i \sim (0, \sigma^2_A)$, $b_i \sim (0, \sigma^2_B)$ and $\epsilon_{ij} \sim (0, \sigma^2_E)$, where a_i, b_i, ϵ_{ij} are mutually independant. And refer to this model as the homogenous random effects model.

8.3 Linear Statistics

We focus on a simple mean

$$\hat{\mu}_x = \frac{1}{N} \sum_i \sum_j Z_{ij} X_{ij} = \frac{1}{N} \sum_{\eta=1}^{N} X_\eta$$

The variance of this mean is (when we allow column and row specific variances $\sigma^2_{A(i)}, \sigma^2_{B(j)}, \sigma^2_{E(i,j)}$):

$$V_{RE}(\hat{\mu_x}) = Var(\frac{1}{N} \sum_i \sum_j Z_{ij} \hat{X}_{ij}) \tag{18}$$

$$= \frac{1}{N^2} Var(\sum_i \sum_j Z_{ij} \hat{X}_{ij}) \tag{19}$$

$$= \frac{1}{N^2} Var\left(\sum_i \sum_j Z_{ij} (\mu + a_i + b_j + \epsilon_{ij}) \right) \tag{20}$$

$$= \frac{1}{N^2} Var\left(\sum_i \sum_j Z_{ij}\mu + Z_{ij}a_i + Z_{ij}b_j + Z_{ij}\epsilon_{ij}) \right) \tag{21}$$

$$= \frac{1}{N^2} \left(Var(\sum_i \sum_j Z_{ij}\mu) + Var(\sum_i \sum_j Z_{ij}a_i) + Var(\sum_i \sum_j Z_{ij}b_j) + Var(\sum_i \sum_j Z_{ij}\epsilon_{ij}) \right) \tag{22}$$

$$= \frac{1}{N^2} \left(0 + \sum_i Var(n_{i\bullet} a_i) + \sum_j var(n_{\bullet j} b_j) + \sum_i Var(\sum_j Z_{ij}\epsilon_{ij}) \right) \tag{23}$$

$$= \frac{1}{N^2} \left(\sum_i n^2_{i\bullet} \sigma^2_{A(i)} + \sum_j n^2_{\bullet j} \sigma^2_{B(j)} + \sum_i \sum_j Z_{ij} \sigma^2_{E(i,j)} \right) \tag{24}$$

Under det homogenous random effects model above, with homogenous variances$(\sigma^2_A, \sigma^2_B, \sigma^2_E)$,

starting from eq. 23 above:

$$V_{HRE}(\hat{\mu}_x) = \frac{1}{N^2}\left(0 + \sum_i Var(n_{i\bullet}a_i) + \sum_j var(n_{\bullet j}b_j) + \sum_i Var(\sum_j Z_{ij}\epsilon_{ij})\right) \tag{25}$$

$$= \frac{1}{N^2}\left(\sum_i n_{i\bullet}^2 Var(a_i) + \sum_j n_{\bullet j}^2 Var(b_j) + \sum_i \sum_j Var(Z_{ij}\epsilon_{ij})\right) \tag{26}$$

$$= \nu_A \frac{\sigma_A^2}{N} + \nu_B \frac{\sigma_B^2}{N} + \frac{\sigma_E^2}{N} \tag{27}$$

(Remember: $\nu_A = \frac{1}{N}\sum_{i=1}^R n_{i\bullet}^2$).

So what is a good estimate of the pooled variance σ_A^2? From the inhomogenous case, (eq.24), we see that the variances is weighted by $n_{i\bullet}^2$. So if there is a systematic difference between the variance of the frequently occuring items and the rare occuring items care must be taken when estimating σ_A^2. Using a pooled estimate of σ_A^2 that weights entities equally would lead to an under-/overestimate of the viance of $\hat{\mu}_x$.

8.4 Bootstrap methods

8.4.1 Naive bootstrap

The usual bootstrap procedure resamples the data i.i.d. from the empirical distribution. So S^* would consist of N rows $(I_\eta^*, J_\eta^*, X_\eta^*)$ drawn independently and uniformly from the N rows of S. So

$$\hat{\mu}_x = \frac{1}{N}\sum_{\eta=1}^N X_\eta^*$$

.

We use the U-statistic to estimate the variance of the Naive Bootstrap estimate of the mean. We look at all pairs of samples, $(X_l, X_{l'})$, and view a function of these, $h(X_l, X_{l'}) = X_l^2 - X_l X_{l'}$ as an unbiased estimates of the variance. But the U-statistic requires that the kernel, h, is symmetric in its arguments, and thus we produce the symmetric version of h as $h'(X_l, X_{l'}) = \frac{1}{2}h(X_l, X_{l'}) + h(X_{l'}, X_l)$. We get $U = \binom{n}{2}^{-1}\sum_{i<j} h'(X_i, X_j) = s_x = \frac{1}{n-1}\sum_{i=1}^N (X_i - \bar{X})^2$

We then get

$$V_{RE}(\hat{\mu}_x) = \frac{1}{2N^3}\sum_{l=1}^N\sum_{l'=1}^N (X_l - X_{l'})^2 \tag{28}$$

$$= \frac{1}{2N^3}\sum_i\sum_j\sum_{i'}\sum_{j'} Z_{ij}Z_{i'j'}(a_i - a_{i'} + b_j - b_{j'} + \epsilon_{ij} - \epsilon_{i'j'})^2 \tag{29}$$

The author then shows that the expectation in the Random Effects model of the variance of the Naive Bootstrap estimate of the mean is (using the U-statistic above):

$$E_{RE}(V_{NB}(\hat{\mu}_x)) = \frac{1}{N^2}\sum_i \sigma^2_{A(i)} n_{i\bullet}\left(1-\frac{n_{i\bullet}}{N}\right)+\frac{1}{N^2}\sum_j \sigma^2_{B(j)} n_{\bullet j}\left(1-\frac{n_{\bullet j}}{N}\right)+\frac{1}{N^2}\sum_i\sum_j Z_{ij}\sigma^2_{E(i,j)} \tag{30}$$

And under the homogenous random effects model(from eq. 30):

$$E_{HRE}(V_{NB}(\hat{\mu}_x)) = \frac{1}{N^2}\sum_i \sigma^2_A n_{i\bullet}\left(1-\frac{n_{i\bullet}}{N}\right)+\frac{1}{N^2}\sum_j \sigma^2_B n_{\bullet j}\left(1-\frac{n_{\bullet j}}{N}\right)+\frac{1}{N^2}\sum_i\sum_j Z_{ij}\sigma^2_E \tag{31}$$

$$= \frac{\sigma^2_A}{N^2}\left(\sum_i n_{i\bullet} - \sum \frac{n^2_{i\bullet}}{N}\right)+\frac{\sigma^2_B}{N^2}\left(\sum_j n_{\bullet j} - \sum_j \frac{n^2_{\bullet j}}{N}\right)+\frac{\sigma^2_E}{N} \tag{32}$$

$$= \frac{\sigma^2_A}{N}\left(1-\frac{\nu_A}{N}\right)+\frac{\sigma^2_B}{N}\left(1-\frac{\nu_B}{N}\right)+\frac{\sigma^2_E}{N} \tag{33}$$

If we compare the results of (33) with (27) we see that for situations where $\nu_A << N$ the variance contribution due to the rows are underestimated by a factor of almost ν_A, or precicely $\frac{\nu_A}{1-\frac{\nu_A}{N}}$. This may be substantial.

8.4.2 Pigeonhole bootstrap

The naive bootstrap fails because it ignores similarities between elements of the same row and/or column. In the Pigeonhole bootstrap we place out data in a $C \times R$ matrix and resample a set of rows and a set of columns, then we take the intersection as our bootstrapped data. The sampling is done with replacement.

Formaly we sample rows r^*_i i.i.d. from $U\{1,...,R\}$ for $i = 1,...,R$ and columns c^*_j i.i.d. from $U\{1,...,C\}$ for $j = 1,...,C$. Rows and columns are sampled independently some number B times, where B is the number of times we resample.

The resampled data set has $Z^*_{ij} = Z_{r^*_i c^*_j}$ and where $Z^*_{ij} = 1$ we have $X^*_{ij} = X_{r^*_i c^*_j}$

Example, let $X = \begin{bmatrix} & X_{12} & & & \\ & & & X_{24} & \\ & X_{32} & & & \\ X_{41} & & X_{43} & X_{44} & \\ & X_{52} & & & X_{55} \end{bmatrix}$

And let $r^* = \{1, 5, 4, 1, 2\}$ and $c^* = \{5, 2, 4, 4, 1\}$, the resample then is

$$X^* = \begin{bmatrix} X^*_{r_1^* c_1^*} & X^*_{r_1^* c_2^*} & X^*_{r_1^* c_3^*} & X^*_{r_1^* c_4^*} & X^*_{r_1^* c_5^*} \\ X^*_{r_2^* c_1^*} & X^*_{r_2^* c_2^*} & X^*_{r_2^* c_3^*} & X^*_{r_2^* c_4^*} & X^*_{r_2^* c_5^*} \\ X^*_{r_3^* c_1^*} & X^*_{r_3^* c_2^*} & X^*_{r_3^* c_3^*} & X^*_{r_3^* c_4^*} & X^*_{r_3^* c_5^*} \\ X^*_{r_4^* c_1^*} & X^*_{r_4^* c_2^*} & X^*_{r_4^* c_3^*} & X^*_{r_4^* c_4^*} & X^*_{r_4^* c_5^*} \\ X^*_{r_5^* c_1^*} & X^*_{r_5^* c_2^*} & X^*_{r_5^* c_3^*} & X^*_{r_5^* c_4^*} & X^*_{r_5^* c_5^*} \end{bmatrix}$$

$$= \begin{bmatrix} X_{15} & X_{12} & X_{14} & X_{14} & X_{11} \\ X_{55} & X_{52} & X_{54} & X_{54} & X_{51} \\ X_{45} & X_{42} & X_{44} & X_{44} & X_{41} \\ X_{15} & X_{12} & X_{14} & X_{14} & X_{11} \\ X_{25} & X_{22} & X_{24} & X_{24} & X_{21} \end{bmatrix}$$

$$= \begin{bmatrix} & X_{12} & & & \\ X_{55} & X_{52} & & & \\ & & X_{44} & X_{44} & X_{41} \\ & X_{12} & & & \\ & & X_{24} & X_{24} & \end{bmatrix}$$

The variance of the total ($T_x = \sum_i \sum_j Z_{ij} X_{ij}$) in pigeonhole bootstrapping is proved to be:

$$V_{PB}(T_x^*) = \left(\frac{1}{RC} - \frac{1}{R} - \frac{1}{C} T_x^2\right) + \left(1 - \frac{1}{C}\right) \sum_i T_{xi\bullet}^2 + \left(1 - \frac{1}{R}\right) \sum_j T_{x\bullet j}^2 + \sum_i \sum_j Z_{ij} X_{ij}^2 \tag{34}$$

Where $T_{xi\bullet} = \sum_j Z_{ij} X_{ij}$ and $T_{x\bullet j} = \sum_i Z_{ij} X_{ij}$.

TODO: do the math

Under the hetrogenous random effects model, the expectation of the variance of the bootstrapped mean is:

$$E_{RE}(V_{PB}(\hat{\mu}_x) \approx \frac{1}{N^2} \left[\sum_i \sigma^2_{A(i)} (n^2_{i\bullet} + 2n_{i\bullet}) + \sum_j \sigma^2_{B(j)} (n^2_{\bullet j} + 2n_{\bullet j}) + 3 \sum_i \sum_j Z_{ij} \sigma^2_{E(i,j)} \right] \tag{35}$$

And finally a simplification to the Expected pigeonhole bootstrap mean under the homogenous random effects model:

$$E_{HRE}(V_{PB}(\hat{\mu}_x)) \approx \frac{1}{N} (\sigma^2_A (\nu_A + 2) + \sigma^2_B (\nu_B + 2) + 3\sigma^2_E) \tag{36}$$

Further on Owen gives an argument for mean consistency under some loose conditions. He defines $\epsilon_N = max\left(\frac{1}{R}, \frac{1}{C}, \frac{\nu_A}{N}, \frac{\nu_B}{N}, \frac{1}{\nu_A}, \frac{1}{\nu_B}, max_i \frac{n_{i\bullet}}{N}, max_j \frac{n_{\bullet j}}{N}\right)$, and shows that under the aforementioned loose conditions:

$$\frac{E(V_{PB}(\hat{\mu}_x)) - V_{RE}(\hat{\mu}_x)}{V_{RE}(\hat{\mu}_x)} = O(\epsilon_N)$$

8.4.3 Netflix example

As an example problem, Owens looks at the day of the week effect in the Netflix dataset. When plotting the average movie ratinge for each day of the week, one can see that the tuesdays have the lowest average, and sunday has the highest. The question is wheter the observed difference is within the variance of the data. The observed values are $\hat{\mu}_{tue} = 3.595808$ and $\hat{\mu}_{sun} = 3.616449$, giving a difference of $\hat{\mu}_{tue} - \hat{\mu}_{sun} = 0.0206$ witch is a small number in the scale of the ratings, but is it also small when compared to the variance. Owens uses Pigeonhole Bootstrapping to estimate the variance of the averages, and finds that the observed difference is about 8 times the standard deviation, concluding that the difference is real.

8.5 Practical Piegonhole Bootstrappping

In summary, a Pigeonhole Bootstrap is based on resampling rows $(r_1^*, ..., r_R^*)$ and collums $(c_1^*, ..., c_C^*)$ from our matrix M, and let our resampled matrix M^* consist of values $m_{ij}^* = m_{r_i^* c_j^*}$ where they exist.

In the Netflix dataset we have 17770 rows and 480189 collums, it is not doable to build the entire matrix in memory, this would require more than 10Gb of memory. So instead we only remember the actual values and their coordinates. We have values $\eta_n = (I_n, J_n, X_n)$, $n \in 1, ..., N$. We sample rows $r_i^* \sim U\{1, ..., R\}$, and collums $c_j^* \sim U\{1, ..., C\}$. We note that the order of the samples are not important when estimating a mean, so we sort our row and collum samples. We also sort our data on row-column form, we can now search sequentially through the data to build our sample. This can now be done in time linear to the data and sample size.

The mean in the Netflix data set is $\hat{\mu} = 3.603308$. An experiment shows the difference in variance estimates:

- A naive bootstrap with 100 resamples gives $var(\hat{\mu}_{NB}) = 1.068177 \times 10^{-8}$
- A Pigeonhole bootstrap with 100 resamples gives $var(\hat{\mu}_{PB}) = 9.720581 \times 10^{-5}$

8.5.1 The use of variance estimates

When shrinking parameters and values the scale is of the essence, and the scale is measured by the variance of the value. If we could find the variance of the global mean and movie and user offsets we could then calculate a confidence intervall for the baseline predictions used in most prediction methods.

In a SVD setting we have

$$X \approx UM,$$

a prediction $\hat{r}_{ij}$ is then the i'th row in U multiplied with the j'th collum in M, so if we know the variances of U and M, we would also be able to produce confidence intervalls for our predictions. The Korbell team uses the term 'confidence score' for an estimate of these values.

8.5.2 Metric Neighbourhood Predictor Bootstrapping

The Metric Neighbourhood Predictor (MNP), is not a parametric model. To estimate the variances of predictions made by MNP's I have trained a large number of MNP's with different dimensions, from 10-50. An estimate of the prediction variance can then be made from simply predicting equal sets of ratings in the different MNP's, and then calculating the empirical variance of the results. This showed that the prediction variance was remarkably low, $\bar{\sigma}_x = 0.00891$, indicating the convergence to a stable solution.

9 Concluding remarks

9.1 About this thesis

I have run hundreds of model fittings with various versions of the methods presented in this thesis, and as is probably normal, I feel the results could have been better and more interesting. But on the other hand I am happy to have been able to produce well functioning models on such a huge dataset. One regret is not putting more time into the implementations of the Logarithmic model presented in section 7.2, the actual implementations was far more difficult than I had anticipated, and more time consuming in terms of running time than the other methods. Still it is in many ways the most interesting model, with clear input and a interpretable output. Also the practical use of Pigeonhole Bootstrapping is regrettably on a minimum, I believe this would prove valuable to most model-like implementations like the SVD implementation presented in section 4.1.

One interesting finding was the KNN by MNP on selected cases. It is very interesting that a method is able to predict where it works well, and with the help of variance estimates on other models it might be possible to build an ensamble of predictors to cover an entire data set. Each predictor given main responsibility over cases where it functions optimaly.

One important piece of workmanship lacking from the implemented models is the removal of global effects and shrinkage of estimated parameters and data. The reason I have not ventured into this is that it sort of breaks the theory behind the models. In for example the SVD implementations you obviously loose the orthogonality of the factors when shrinking the estimated parameters, and while this helps remedy overfitting, it is not so easy to understand how the result should be interpreted. Of course most of the competitors in the Netflix Prize do not care so much for the theory, they aim to produce the best possible predictions on a very specific set. The Korbell team, who have deep theoretical foundations, view shrinkage as a continous alternative to parameter selection, and while this works well, it does make interpretation more difficult.

9.2 The convergence of methods

An interesting point is that our best efforts in creating recommender systems are only slightly better than the most naive ways. The leader on the Netflix Prize are about 20% better, measured by rmse, than just predicting every rating as the movie's average rating. This does not sound good, but the implications of a 20% increase in prediction accuracy should not be understated. The quality

of a top 100 list of movies, compiled to a specific user is hugely affected by this increase. The Korbell team discusses this in their paper [3] where they estimate the probabilities of a movie known to be well liked by the user to end up on a top list. They show that even small increases in the rmse measure result in largely increased probabilities for the known good movie to end up on a user's top list. On the other hand, one could ask how far the increase in prediction accuracy can go when all current methods seem to converge on about 15% increase on the naive method. Mixture models take it a bit further, towards the 20% mark, but is this a limit? There are of course limits to how well a human being can be predicted to behave, even the most deterministic among us realize that we can not be modeled completly, but wheter we are close to the limit remains an open question. Another perspective is that very different models seem to converge to the same accuracy, Restricted Boltzmann Machines, Matrix Factorisation, Neighbourhood based methods, what do they have in common? One common denominator in the Netflix Price is of course the information available to the models. Perhaps this is as far as we can go with the information available. Some competitors have seen increases in their results when using information derived from the title of the movies, others have merged in external data with modest positive effects. But none of the contestants have reported significantly increased results even when pouring in data from the International Movie Database (IMDB.com) or others. Another common denominator is the domain, our rating of movies are affected by external noise, like how was life at work the day you rated 'The Office', did the news report on the fate of diamond smugglers the day you rated 'Blood Dimond', where you in love when watching 'Notting Hill'. This information is not part of the rating set, and must be considered as noise. It is entirely possible that the effects not accounted for by the presented mehtods are down to noise, but it is also possible that some quantum leap will increase the prediction accuracy by another 10%.

9.3 The future of recommender systems

The industry of recommender systems is growing rapidly, and the pace will pick up even further in the years to come. While writing this thesis I have subscribed to a number of different services using recommender systems as their basic tool. The results are somewhat impressive:

- Amazon gives conservative recommendations about books I should buy two times a week. The recommendations are conservative in the way of staying within the genre of books I have already bought. The quality is high, meaning that I seriously consider a large number of the recommended

books, and ended up bying among others "Programming Collective Intelligence", just what I needed for this thesis...

- iRead (in Facebook) recommends literature based on what I have reported to have read. Again the results are impressive, they select about 10 books each week from a repository of more than 7 million books. When the results actually is interesting to me, they must have done a good job.

- Match.com is an online dating service. I completed a long tedious questionare, including questions on how long my index finger was relative to my ring finger, and other unexpected questions. Twice a week I receive 12 profiles of girls they claim match my profile. Interestingly, all recommended profiles are girls who are younger than myself, can it be my long ring finger? I have regrettably not had the oppurtunity to investigate the quality of these recommendations.

The domain of the recommender systems increases slowly, lots of work has been done on the simple user-product model where the products need to be reasonably similar to one another. Amazon does not recommend candy bars or gardening equipment although they sell them in the same online store, the recommended product must be in the same domain as the purchased product.

When eSmak was conceived the idea was not just to recommend movies, but slowly expand to books, wine and other domains. We hoped to be able to use the knowledge we had on a user's movie taste to recommend a good red wine to go with the movie. This cross domain link is not unfeasible, some directions in cognitive psychology suggest that there are underlying representational constructs in our brains that have influence on different every day domains. These constructs can be as basic as how do we understand color? Does this have anything to do with how we interpret taste? Peter Gärdenfors presents what he calls Conceptual Spaces in his book "Conceptual Spaces: The Geometry of Thought" [7]. The basic idea is that thought can be represented as manipulation of states in geometric spaces. The spaces in question is built up of domains, like the aforementioned color domain. The color domain is a geometric space built up of the traditional color disc where the hue is represented on the perimeter and the saturation is proportional to the distance from the center, crossed with a dimension representing luminescence, from absolute darkness to maximum light, see figure 23.

Gärdenfors claims this can explain concepts like skin-tone. In the previous century, Europeans labeled American Indians as red skins, Europeans as white, Asians as yellow etc. But when looking at the skin of a asian, it is far from yellow,

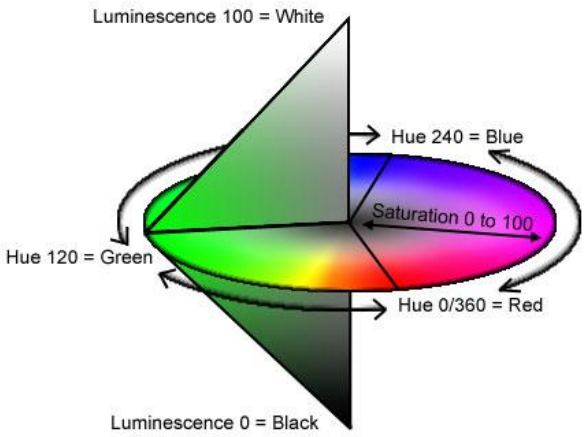

Figure 23: HLS-Color space

so where does this label come from? Gärdenfors argues that if we map out the total space of skin colours, it will have the same geometry as the entire colour space, but be confined to a small subset of the available colours. We, as human beings, lend the terminology of the color space and apply it to the subspace of skin colours. So yellow when reffering to skin tones means in the direction of yellow when extrapolated to the full space. Also the dimensions of the color space is correlated, one can not talk about something that is very red as well as very dark, so darkness is sort of negatively correlated with saturation. Other meanings of everyday language is also borrowed from the familiar colorspace, like light and dark emotions/humor/literature, they are all analogies to the luminecence of the colorspace.

One could speculate and view matrix factorization as a first attempt to understand the human cognition in a geometric way, the ultimate goal being to build a full conceptual model of each user by means of the available information. We looked at how the different factors in the SVD mapped to different movies, concluding that the factors seem to map onto everyday concepts, although not easily termed.

So if we let ourself believe a little bit that a recommender system could model parts of the conceptual configuration of the users, how will the future of recommender systems look?

Given Gärdenfors' Conceptual Spaces, the understanding of interdomain correlations will help bring down the barriers, your preferences in books is relevant to your preferences in movies, music and even wine and gardening equipment. When the barriers come down, the amount of available information goes up,

the grocery you have purchased using a discount card or coupon will contribute information to what movie should be recommended to you. So when you return from the grocery store, having bought popcorn and Coca Cola, the recommender system on you pay-per-view tv channel will not recommend a documentary on Holocost, but rather a family of children movie.

Understanding a problem domain also offers control of the domain. The ones among us with inclination to conspiracy theories will have a feast. When privatly/governmently owned media shows you just what you want to see or hear or read, how will this affect your view of the world. Who decides your view of the world? When searching on the future Google for terms like 'war', and the company War.inc has bought the adword 'war', and on the returned page they present their view of the world, matced to the political/etical views they belive you have... ok, far fetched. The point is that increased understanding of how people's preferences work also increases the control of your preferences for those willing to pay.

Whether one pays any credit to Conceptual Spaces or not, the future of recommendation systems seems to be in breaking the interdomain barriers. Recommender systems will, no doubt, be ever more present in our interaction with commerce and other aspects of life.

References

[1] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. 2007.

[2] Robert M. Bell, Yehuda Koren, and Chris Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. 2007.

[3] Robert M. Bell, Yehuda Koren, and Chris Volinsky. Factorization meets the neighborhood: a multifaceted collaborative filtering model. *Netflix*, 2008.

[4] Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.

[5] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine, 1998.

[6] Rob Brown. Evolution and wisdom of crowds, 2007. [Online; accessed 8-11-2008].

[7] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. Bradford Books MIT Press, 2000.

[8] Knut Hegna. Universell bibliografisk kontroll mål, midler, teknologi, 2002. [In Norwegian].

[9] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 1999.

[10] Art B. Owen. The pigeonhole bootstrap. 2007.

[11] Sam Roweis. Em algorithms for pca and spca. *Advances in Neural Information Processing Systems 10*.

[12] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, 2007.

[13] G. Salton. *Automatic Text Processing*. Addison-Wesley, 1989.

[14] Robert E. Schapire. The boosting approach to machine learning. an overview. *MSRI Workshop on Nonlinear Estimation and Classification, 2002*, 2001.

[15] James Surowiecki. *The Wisdom of Crowds*. Abacus, 2004.

[16] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. On the gravity recommendation system. 2007. http://home.mit.bme.hu/ gtakacs/download/gravity.pdf.

[17] Brandyn Webb. Netflix update: Try this at home, 2007. [Personal blog at http://sifter.org/ simon/journal/20061211.html under the name Simon Funk].

[18] Wikipedia. Information filtering system — wikipedia, the free encyclopedia, 2008. [Online; accessed 7-August-2008].

[19] Wikipedia. Singular value decomposition — wikipedia, the free encyclopedia, 2008. [Online; accessed 22-January-2008].

A SVD, top and bottom movies of each feature.

Maximums	Minimums
Mobsters and Mormons(2005)	Six Feet Under(2003)
Hockey Mom(2004)	Seinfeld(1992)
Larryboy and the Rumor Weed(1999)	The Simpsons(1992)
The Land Before Time IV(1996)	Firefly(2002)
Love on Lay-Away(2004)	The Sopranos(2001)
Dune(1984)	Six Feet Under(2001)
The Land Before Time VI(1998)	24(2002)
Bram Stoker's: To Die For(1989)	The Sopranos(2002)
My Wife's Murder(2005)	The Sopranos(2000)
Curious(2005)	Dead Like Me(2004)
Journey Into Amazing Caves(2001)	The Simpsons(1993)
Predator Island(2005)	The Simpsons(1994)
The Triangle(2005)	Seinfeld(1991)
Avia Vampire Hunter(2005)	House(2004)
Dark Harvest 2(2004)	Six Feet Under(2004)
Ax 'Em(2002)	The Shawshank Redemption(1994)
Hazaaron Khwaishein Aisi(2003)	Anne of Green Gables(1987)
Zodiac Killer(2004)	Band of Brothers(2001)
Absolution(2003)	Ken Burns' Civil War(1990)
The Worst Horror Movie Ever Made(2005)	The West Wing(1999)
Vampire Assassins(2005)	The West Wing(2001)
Death Mask(1998)	The Sopranos(2004)
Half-Caste(2004)	The West Wing(2002)
Alone in a Haunted House(2004)	Lord of the Rings(2002)
Expo(2005)	Veronica Mars(2004)
Drive In(2001)	Lord of the Rings(2003)
The Horror Within(2005)	The Lord of the Rings(2001)

Figure 24: Feature 1, higest and lowest movies.

Maximums	**Minimums**
Lost in Translation(2003)	Runaway Bride(1999)
The Royal Tenenbaums(2001)	Dungeons and Dragons(2000)
Eternal Sunshine of the Spotless Mind(2004)	Crocodile Dundee in Los Angeles(2001)
Dogville(2004)	Exit Wounds(2001)
Punch-Drunk Love(2002)	Jack Frost(1998)
The Life Aquatic with Steve Zissou(2004)	Big Momma's House(2000)
Before Sunset(2004)	What Women Want(2000)
Napoleon Dynamite(2004)	Congo(1995)
Adaptation(2002)	Weekend at Bernie's 2(1993)
Sideways(2004)	Cop and a Half(1993)
Primer(2004)	Rocky V(1990)
Fahrenheit 9/11(2004)	Home Alone 3(1997)
Sin City(2005)	S.W.A.T.(2003)
Memento(2000)	Van Helsing(2004)
Being John Malkovich(1999)	Batman and Robin(1997)
I Heart Huckabees(2004)	Fire Down Below(1997)
Pulp Fiction(1994)	Collateral Damage(2002)
American Beauty(1999)	The Glimmer Man(1996)
The Mother(2003)	Gone in 60 Seconds(2000)
Garden State(2004)	Speed 2(1997)
Kill Bill(2003)	Eddie(1996)
Oldboy(2003)	Miss Congeniality(2000)
A Clockwork Orange(1971)	Look Who's Talking Now(1993)
Closer(2004)	The Fast and the Furious(2001)
Brothers(2005)	Maid in Manhattan(2002)
Shaun of the Dead(2004)	Armageddon(1998)
Intermission(2004)	On Deadly Ground(1994)
Annie Hall(1977)	The Wedding Planner(2001)
Intimate Strangers(2004)	Coyote Ugly(2000)
Secretary(2002)	Pearl Harbor(2001)

Figure 25: Feature 2, highest and lowest scoring movies

Maximums	**Minimums**
House of 1(2003)	One Tree Hill(2003)
Fear and Loathing in Las Vegas(1998)	Will and Grace(2000)
Freddy Got Fingered(2001)	Star Trek(2000)
Orgazmo(1998)	Star Trek(1996)
Super Troopers(2002)	Star Trek(1998)
Jay and Silent Bob Strike Back(2001)	Star Trek(1997)
Wake Up(2004)	Dawson's Creek(1998)
The Texas Chainsaw Massacre 2(1983)	Dawson's Creek(2001)
Club Dread(2004)	Felicity(1999)
Wet Hot American Summer(2001)	Dawson's Creek(1998)
Friday the 13th(1988)	Dawson's Creek(2003)
Jackass(2002)	The Best of Friends(1994)
Half Baked(1998)	The Best of Friends(1994)
Natural Born Killers(1994)	Friends(2002)
Friday the 13th(1989)	Friends(1996)
BASEketball(1998)	The Best of Friends(1994)
Child's Play 2(1990)	Friends(1994)
Friday the 13th(1984)	Dawson's Creek(2000)
Little Nicky(2000)	Dawson's Creek(1999)
The Rules of Attraction(2002)	Friends(1999)
Beavis and Butt-head Do America(1996)	Friends(2004)
Friday the 13th(1982)	The Best of Friends(1994)
Jason Goes to Hell(1993)	The Best of Friends(1994)
Freddy vs. Jason(2003)	Friends(2001)
May(2003)	Friends(1999)
Evil Dead 2(1987)	The Best of Friends(1994)
Scary Movie 2(2001)	The Best of Friends(1997)
Killer Klowns from Outer Space(1988)	The Best of Friends(1996)
Four Rooms(1995)	Friends(1997)
Friday the 13th(1981)	Friends(1998)

Figure 26: Feature 3, highest and lowest scoring movies

Maximums	**Minimums**
Luster(2002)	Jaws(1975)
Battle Athletes Victory(1999)	The Matrix(1999)
Cardcaptor Sakura(1999)	Ferris Bueller's Day Off(1986)
The Last Year(2002)	A Few Good Men(1992)
Boys Briefs(2000)	Beverly Hills Cop(1984)
Best of Boys in Love(2000)	Indiana Jones and the Last Crusade(1989)
Mariah Carey(1999)	Good Will Hunting(1997)
No Ordinary Love(1994)	Dances With Wolves(1990)
Very Natural Thing(1973)	Pulp Fiction(1994)
Glitter(2001)	Star Wars(1983)
Gone But Not Forgotten(2003)	Jurassic Park(1993)
Denied(2004)	National Lampoon's Vacation(1983)
Battle Athletes Victory(1998)	Raiders of the Lost Ark(1981)
A Real Young Girl(2001)	Saving Private Ryan(1998)
Amazing Nurse Nanako(2000)	Terminator 2(1991)
Drift(2001)	Patriot Games(1992)
The Company(2003)	The Patriot(2000)
Angel Sanctuary(2000)	National Lampoon's Animal House(1978)
In the Flesh(1997)	Gladiator(2000)
'N Sync(1999)	The Hunt for Red October(1990)
Under the Cherry Moon(1986)	The Terminator(1984)
Trois(2000)	Top Gun(1986)
Spice World(1998)	Caddyshack(1980)
Battle Athletes Victory(1999)	Lethal Weapon(1987)
Slayers Try DVD Collection(1997)	Michael Moore Hates America(2004)
Battle Athletes Victory(1998)	Die Hard(1988)
Pokemon(1999)	Rocky(1976)
BoyFriends(2000)	Gladiator(2000)
Boys Life 4(2003)	Forrest Gump(1994)
Gang of Roses(2003)	Braveheart(1995)

Figure 27: Feature 4, highest and lowest scoring movies

Maximums	Minimums
Queer as Folk(2002)	WWE(2004)
Angels in America(2003)	The Three Stooges(1937)
Queer as Folk(2001)	The Matrix(1999)
Camp(2003)	NFL(2005)
A Home at the End of the World(2004)	Rush(1991)
Queer as Folk(2003)	L'Eclisse(1962)
The Hours(2002)	Kill!(1969)
Sordid Lives(2001)	Godsmack(2002)
The Piano(1993)	NFL(2004)
If These Walls Could Talk 2(2000)	WWE(2005)
The Broken Hearts Club(2000)	The Three Stooges(1943)
Sophie's Choice(1982)	Johnny Cash(2003)
The Adventures of Priscilla(1994)	SpongeBob SquarePants(2002)
If These Walls Could Talk(1996)	Sarfarosh(1999)
Far from Heaven(2002)	The Beverly Hillbillies(1962)
Bad Education(2004)	The Man Show Boy / Household Hints from Adult Film Stars(2005)
The Laramie Project(2002)	Fox and His Friends(1975)
I'm the One That I Want(2000)	Masters of Poker(2005)
Muriel's Wedding(1994)	ECW(2001)
Being Julia(2004)	The Adventures of Ford Fairlane(1990)
Funny Girl(1968)	Godannar(2005)
Vera Drake(2004)	The Three Stooges Double Feature(1947)
Annie Hall(1977)	The Matrix(2003)
The Crying Game(1992)	PRIDE Fighting Championships(2002)
De-Lovely(2004)	Red Dwarf(1988)
Iris(2001)	FahrenHYPE 9/11(2004)
Bowling for Columbine(2002)	Benny Hill(1969)
Tea with Mussolini(1999)	Sports Illustrated Swimsuit Edition(2002)
Before Night Falls(2000)	Michael Moore Hates America(2004)
Hannah and Her Sisters(1986)	Celsius 41.11(2004)

Figure 28: Feature 5, highest and lowest scoring movies

Maximums	**Minimums**
The Twilight Zone(1964)	House of Sand and Fog(2003)
The Twilight Zone(1963)	Secret Window(2004)
The Twilight Zone(1963)	All I Want(2002)
The Twilight Zone(1961)	The Upside of Anger(2005)
The Three Stooges(1947)	Alexander(2004)
The Twilight Zone(1964)	The Clearing(2004)
Dragon Ball Z(1998)	Tiptoes(2003)
Dragon Ball Z(2000)	The Ladykillers(2004)
The Twilight Zone(1960)	Eye of the Beholder(2000)
The Twilight Zone(1960)	Monster's Ball(2001)
Dragon Ball Z(1989)	Open Water(2004)
The Twilight Zone(1963)	Closer(2004)
The Twilight Zone(1961)	Stateside(2004)
The Twilight Zone(1964)	City of Ghosts(2003)
The Twilight Zone(1963)	Wicker Park(2004)
The Twilight Zone(1968)	Gangs of New York(2002)
The Three Stooges(1938)	People I Know(2003)
The Twilight Zone(1964)	Birth(2004)
The Twilight Zone(1962)	Secret Things(2002)
The Three Stooges(1943)	Alexander(2004)
The Twilight Zone(1967)	The Final Cut(2004)
The Three Stooges(1940)	A.I. Artificial Intelligence(2001)
The Twilight Zone(1963)	Suspect Zero(2004)
The Twilight Zone(1964)	The Village(2004)
Dragon Ball Z(1992)	The Door in the Floor(2004)
The Twilight Zone(1963)	The Human Stain(2003)
Doctor Who(1976)	The Jacket(2005)
Scooby-Doo's Creepiest Capers(2000)	Chrystal(2004)
The Twilight Zone(1960)	Vanilla Sky(2001)
The Twilight Zone(1962)	In the Cut(2003)

Figure 29: Feature 6, highest and lowest scoring movies

Maximums	**Minimums**
Pirates of the Caribbean(2003)	Citizen Kane(1941)
Moulin Rouge(2001)	The Thin Red Line(1998)
Moulin Rouge(2001)	Body Heat(1981)
Love Actually(2003)	Glengarry Glen Ross(1992)
Elf(2003)	Raging Bull(1980)
The Hitchhiker's Guide to the Galaxy(2005)	Damn the Defiant(1962)
Dodgeball(2004)	Who'll Stop The Rain(1978)
Friends(1995)	Rambo(1982)
Pirates of the Caribbean(2003)	The Postman Always Rings Twice(1981)
Saved!(2004)	Gummo(1997)
Harry Potter and the Sorcerer's Stone(2001)	Shane(1953)
50 First Dates(2004)	Chinatown(1974)
Bride and Prejudice(2004)	The French Connection II(1975)
The Incredibles(2004)	Quest for Fire(1982)
Garden State(2004)	Rosemary's Baby(1968)
Sex and the City(2000)	Above the Law(1988)
Coupling(2000)	Easy Rider(1969)
Mean Girls(2004)	A Clockwork Orange(1971)
Sex and the City(1999)	Dirty Harry(1972)
Sex and the City(2001)	Death Wish(1974)
Down With Love(2003)	The Birth of a Nation(1915)
Sex and the City(1998)	Last Tango in Paris(1972)
Buffy the Vampire Slayer(1997)	Midnight Cowboy(1969)
Bridget Jones(2004)	The French Connection(1971)
Finding Neverland(2004)	Taxi Driver(1976)
Straight-Jacket(2004)	Apocalypse Now(1979)
Anchorman(2004)	Death Wish 5(1994)
Napoleon Dynamite(2004)	2001(1968)
13 Going on 30(2004)	The Deer Hunter(1978)
Harry Potter and the Prisoner of Azkaban(2004)	Deliverance(1972)

Figure 30: Feature 7, highest and lowest scoring movies

Maximums	Minimums
The Simple Life(2003)	Star Trek VI(1991)
Urban Cowboy(1980)	Star Trek V(1989)
Cocktail(1988)	Stargate SG-1(1998)
Meet the Parents(2000)	Star Trek(1991)
Love Story(1981)	Babylon 5(1994)
Tommy Boy(1995)	Stargate SG-1(2000)
Madonna(1991)	Brazil(1985)
American Pie(1999)	Star Trek(1995)
Less Than Zero(1987)	Star Trek(1993)
There's Something About Mary(1998)	The 13th Warrior(1999)
Indecent Proposal(1993)	Star Trek(1997)
St. Elmo's Fire(1985)	Henry V(1989)
Dirty Dancing(1987)	Luther(2003)
National Lampoon's Christmas Vacation(1989)	Seven Samurai(1954)
Mommie Dearest(1981)	THX 1138(1971)
Big Daddy(1999)	Ran(1985)
The Blue Lagoon(1980)	Russian Ark(2002)
Dragon Ball Z(2003)	Star Trek III(1984)
Boomerang(1992)	Star Trek(1994)
Dragon Ball(1995)	Star Trek(2002)
There's Something About Mary(1998)	Stargate SG-1(1999)
Curb Your Enthusiasm(2000)	Yojimbo(1961)
National Lampoon's Vacation(1983)	The Chronicles of Riddick(2004)
Cops(2004)	Star Trek(1987)
The Toy(1982)	Star Trek(1998)
Jerry Maguire(1996)	The Hitchhiker's Guide to the Galaxy(1981)
Dragon Ball(2003)	Star Trek(1989)
About Last Night...(1986)	Stargate SG-1(1997)
Very Bad Things(1998)	Star Trek(1996)
Grease(1978)	Star Trek(1967)

Figure 31: Feature 8, highest and lowest scoring movies

Maximums	Minimums
Dragon Ball Z(2003)	The Rocky Horror Picture Show(1975)
Dragon Ball Z(2003)	Kung Pow(2002)
One Tree Hill(2003)	Throw Momma From the Train(1987)
Curb Your Enthusiasm(2000)	Elvira(1988)
Dragon Ball GT(2003)	Super Mario Bros.(1993)
Dragon Ball Z(1993)	Drop Dead Fred(1991)
Dragon Ball Z(2000)	Making Mr. Right(1987)
Dragon Ball Z(1992)	2002 Olympic Figure Skating Exhibition(2002)
Viva La Bam(2003)	Hudson Hawk(1991)
UFC 47(2004)	Coneheads(1993)
Dragon Ball Z(1989)	Mars Attacks!(1996)
Dragon Ball Z(2003)	Beetlejuice(1988)
I Can Do Bad All By Myself(2002)	Haunted Honeymoon(1986)
Dragon Ball Z(2002)	The Rocky Horror Picture Show(1975)
Dragon Ball Z(2003)	Ivan Vasilievich(1973)
UFC 49(2004)	The Secret Garden(1975)
Dragon Ball Z(2001)	Addams Family Values(1993)
Dragon Ball Z(2003)	Soapdish(1991)
Dragon Ball Z(2000)	Mixed Nuts(1994)
One Tree Hill(2004)	A.I. Artificial Intelligence(2001)
The Shield(2004)	The Butcher's Wife(1991)
Viva La Bam(2004)	The Addams Family(1991)
Dragon Ball Z(2000)	Death Becomes Her(1992)
Dragon Ball(2002)	Little Shop of Horrors(1986)
Dragon Ball Z(2003)	Moulin Rouge(2001)
Madea's Class Reunion(2003)	My Stepmother is an Alien(1988)
Newlyweds(2003)	Tank Girl(1995)
Friends(1998)	Toys(1992)
Meet the Browns(2004)	Earth Girls Are Easy(1989)
The Best of Friends(1997)	Joe Versus the Volcano(1990)

Figure 32: Feature 9, highest and lowest scoring movies

Maximums	**Minimums**
Dave Matthews Band(1999)	The Exorcist(1973)
Grateful Dawg(2000)	Buffy the Vampire Slayer(1999)
Michael Moore Hates America(2004)	Cabin Fever(2003)
The Work and the Glory(2004)	Buffy the Vampire Slayer(1997)
With Honors(1994)	Final Destination(2000)
Run Ronnie Run(2002)	The Ring(2002)
Where the Buffalo Roam(1980)	Scream 3(2000)
Fierce Creatures(1997)	Final Destination 2(2003)
Coupling(2000)	Alien(1979)
Voices of Iraq(2004)	I Know What You Did Last Summer(1997)
Hudson Hawk(1991)	Monster(2003)
Mating Habits of the Earthbound Human(1999)	Kill Bill(2003)
Sgt. Bilko(1996)	Xena(2001)
Fletch(1985)	Friday the 13th(1980)
Love Affair(1994)	Buffy the Vampire Slayer(2001)
Forget Paris(1995)	Scream 2(1997)
Greedy(1994)	Halloween(1978)
The Man Who Knew Too Little(1997)	The Lost World(1997)
Power of One(1993)	Jaws(1975)
On Any Sunday(1971)	Jurassic Park III(2001)
Blue in the Face(1995)	The Exorcist(1973)
My Fellow Americans(1996)	The Blair Witch Project(1999)
Fletch Lives(1989)	Carrie(1976)
Faster(2003)	Jeepers Creepers 2(2003)
Human Traffic(2000)	Scream(1996)
Oscar(1991)	Open Water(2004)
Hopscotch(1980)	Titanic(1997)
Spies Like Us(1985)	Jeepers Creepers(2001)
Snatch(2000)	The Grudge(2004)
Beautiful Girls(1996)	A Nightmare on Elm Street(1984)

Figure 33: Feature 10, highest and lowest scoring movies

B R-script for logaritmic Maximum Likelihood Estimation.

Not quite upto date...

```
dat <- read.csv("<FILENAME>", header=TRUE, dec=".", sep=";");
ratings <- dat$rating;
users <- as.numeric(levels(factor(dat$userId)))  #1901 users
movies <- as.numeric(levels(factor(dat$movieId))) #848 movies
ratingsArr <- matrix(nrow=length(users), ncol=length(movies));
N <- length(dat$rating);
for(i in 1:N) {
  ratingsArr[dat$userId[i],dat$movieId[i]] <- dat$rating[i];
}

moviesByUser <- split(dat$movieId, factor(dat$userId)) #Get movies by using moviesByUse[toStrin
usersByMovie <- split(dat$userId, factor(dat$movieId))

RATING_MAX <- 6;
all_ratings <- 1:RATING_MAX;
length(users);
length(movies);

ratingDist <- function(x1,x2, theta) {
  (abs(x1-x2)^theta$alpha)/theta$alpha
}

userDist <- function(i,j, theta) {
  ratingDists <- ratingDist(ratingsArr[i,],ratingsArr[j,], theta);
  (1/length(s))*sum(ratingDists[ratingDists > 0], na.rm=TRUE);
}

beta <- function(Dxy, dij, theta) {
  theta$b0 + Dxy*theta$bD+dij*theta$bd+Dxy*dij*theta$bDd;
}
probX_ki_j <- function(j,x,k,i, theta) {
  xjk <- ratingsArr[j,k];
  userDistValue <- userDist(i,j,theta);
  exp(beta(ratingDist(x,xjk,theta), userDistValue ,theta))/sum(exp(beta(ratingDist(all_ratings,
```

```
}

probX_ki <- function(x, k, i, theta) {
  users_bm <- usersByMovie[[toString(k)]];
  users_bm <- users_bm[users_bm != i]
  dim(users_bm) <- c(length(users_bm),1);
  (1/length(users_bm))*sum(apply(users_bm, 1, probX_ki_j, x,k,i,theta),na.rm=TRUE);
}

ll_i <- function(userId, theta) {
  moviesFU <- moviesByUser[[toString(userId)]];
  probs <- rep(NA, length(moviesFU));
  for(k in 1:length(moviesFU)) {
    probs[k] <- probX_ki(ratingsArr[userId,moviesFU[k]], moviesFU[k], userId, theta);
  }

  sum(log(probs[is.na(probs)==0]))
}

ll <- function(rho) {
  theta <- list(alpha=rho[1], gamma=rho[2], b0=rho[3], bD=rho[4], bd=rho[5], bDd=rho[
  print(theta);
  probsByUser <- rep(0, length(users));
  for(i in 1:length(probsByUser)) {
    probsByUser[i] <- ll_i(i, theta);
    print(paste(i, ':',probsByUser[i]));
  }
  -sum(probsByUser);
}

res_nlm <- nlm(ll, c(2,1,1,0.1,0.1,0.1), print.level=1, hessian=FALSE);
summary(res_nlm)

##Weights
wij <- function(i,j,theta) {
theta$wij[max(i,j), min(i,j)];
}
```

```
##Create the weights.  Run every time theta chages.
createwij <- function(theta) {
sumW <- length(users)*theta$gamma+length(ratings);
wijArr <- matrix(ncol=length(users), nrow=length(users));
for(i in 1:length(users)) {
for(j in 1:i) {
moviesI <- moviesByUser[[toString(i)]];
moviesJ <- moviesByUser[[toString(j)]];
wijArr[i,j] <- (theta$gamma+length(intersect(moviesI,moviesJ)))/sumW;
}
}
theta$wij <- wijArr;
}
```