Collaborative Filtering using Temporal Analysis

Abhijith Chandraprabhu

Linköpings universitet
TEKNISKA HÖGSKOLAN

Collaborative Filtering using Temporal Analysis

Department of Mathematics

Abhijith Chandraprabhu

LiTH-MAT-EX--YY/XXXX--SE

Handledare: **Berkant Savas**
                MAI, Linköpings universitet

Examinator: **Lars Eldén**
                MAI, Linköpings universitet

Linköping, 19 September, YYYY

Avdelning, Institution	**Datum**
Division, Department	Date
Division of Applied Mathematics Department of Mathematics Linköpings universitet SE-581 83 Linköping, Sweden	YYYY-09-19

Titel
Title

Collaborative Filtering using Temporal Analysis
Svenk Titel

Författare Abhijith Chandraprabhu
Author

Sammanfattning
Abstract

In todays digital world Information in the form of large scale data has become a very important source of knowledge, this is largely owed to the advances in Data Sciences. One of the recent applications of data mining is to find meaningful and useful relationship between groups of entities, for example between users and movies as in the case of NETFLIX. With the NETLIX competition, the field of Recommender Systems which helps in presenting users with relevant items based on the history rather than an explicit search query, has generated great interest from researchers and this work is one such attempt. When we talk about users' preferences it is natural that it evolves, hence in this work we have tried to account for a new dimension, i.e time, while building the model. On the lines on the NETFLIX competition, the task of predicting preferences involves the ratings of the movies given by users and it is the value of this rating which we try to predict. On the lines on the NETFLIX competition, the task of predicting preferences involves the ratings of the movies by users and it the value of this rating which we try to predict. The data upon which the algorithm is built consists of around 1 million ratings given by around half a million users to around 18k movies. Collaborative Filtering techniques have been employed, where based upon a model which has information of users and movies we try to fetch prediction of new ratings by factoring the user space and movie space and finding a relationship between the required pair. Matrix factorization is done based on Singular Value Decomposition, and owing to the inherent nature of SVD to handle sparse data inefficiently an alternate method based on Alternate Least Squares was used.

Nyckelord
Keywords   Netflix, Recommender Systems, RMSE, Matrix factorization, Sparse SVD, ALS-WR, Low-Rank Approximation,

Abstract

In todays digital world Information in the form of large scale data has become a very important source of knowledge, this is largely owed to the advances in Data Sciences. One of the recent applications of data mining is to find meaningful and useful relationship between groups of entities, for example between users and movies as in the case of NETFLIX. With the NETLIX competition, the field of Recommender Systems which helps in presenting users with relevant items based on the history rather than an explicit search query, has generated great interest from researchers and this work is one such attempt. When we talk about users' preferences it is natural that it evolves, hence in this work we have tried to account for a new dimension, i.e time, while building the model. On the lines on the NET-FLIX competition, the task of predicting preferences involves the ratings of the movies given by users and it is the value of this rating which we try to predict. On the lines on the NETFLIX competition, the task of predicting preferences involves the ratings of the movies by users and it the value of this rating which we try to predict. The data upon which the algorithm is built consists of around 1 million ratings given by around half a million users to around 18k movies. Collaborative Filtering techniques have been employed, where based upon a model which has information of users and movies we try to fetch prediction of new ratings by factoring the user space and movie space and finding a relationship between the required pair. Matrix factorization is done based on Singular Value Decomposition, and owing to the inherent nature of SVD to handle sparse data inefficiently an alternate method based on Alternate Least Squares was used.

Acknowledgments

Parents, Berkant Savas, Lars Elden, Niklas Ekvall, Opposition person.

Contents

Chapter 1

Introduction

In this chapter we introduce the field of Datamining and its relevance. We also introduce the area of research in which this thesis work is done.

1.1 Introduction

The field of Data Mining aka Knowledge Discovery in Data, or KDD saw its first light during the 1990s, and has evolved along with the processing power and storage capacities of modern computer systems. According to IBM, data is being ubiquitously generated at a rate of 2.5 billion gigabytes each day [3]. The amount of data is exploding with as estimate of 35 zetabytes by the year 2020, which seems realistic with just social ineraction websites like twitter contributing around 7 terabytes(TB) each day and facebook generating around 10 terabytes [18]. The analytics possibilities with this vast amounts of data is again as diverse as the sources of data, in the political arena, Big data technology played an important factor for Barack Obama's 2012 presidential campaign, where information from previous election were used [1]. Researchers from Uppsala Monitoring Centre have been able to create a comprehensive database of Adverse Drug Reactions(ADRs) using advanced data mining techniques [4]. In the business front, enables new business intelligence dimension, NETFLIX is using its user ratings information to provide better predictions and thereby helping its customers make choices.

We intend to obtain/infer useful knowledge from raw data, by observing underlying patterns which characterizes the system represented by the raw data. In this complex process of infering knowledge from raw data, data mining forms the part which requires a mathematical approach. The data mining stage is preceded by certain engineering stages like creation and preparation of data. It is succeeded by certain business or domain specific knowledge inference and analysis stages. There are six main tasks within the gamut of data mining, enlisted below, [8]

Classification is learning a function that maps a data item into one of several predefined classes.

Regression is learning a function that maps a data item to a real-valued prediction variable.

Clustering is a common descriptive task where one seeks to identify a finite set of categories or clusters to describe the data.

Summarization involves methods for finding a compact description for a subset of data.

Dependency Modeling consists of finding a model that describes significant dependencies between variables.

Change and deviation detection focuses on discovering the most significant changes in the data from previously measured or normative values.

From a mathematical perspective data mining or pattern recognition can be viewed as a data model fitting problem. The processed data what we use to build a model upon is called as the *training data*. This training data is prepared from the factual previous knowledge. Next after building a model, we would like to ensure its validity. There needs to be some kind of measurement to esnure certain desirable degree of correctness of the model. This forms the *error analysis*, and metrics like *RMSE* are used often. Linear algebra proves to be pivotal in data mining, [7] to build these models.

In this master thesis work, our goal is to build a recommender system, which will help people choose movies they would like to watch. For building the same, we need some previous knowledge, an idea or some methodology on how to model this data, and some measure for degree of accuracy of the model.

The kind of prediction we are dealing with is ratings prediction, which gives an indication of user's preference to items. Depending on the predicted rating recommendations are made, this involves different tasks the most significant of them being matrix factorization and rank reduction. It is very common while dealing with preferences that they tend to change with time, the work by [13] models the data drifting with time. It is important to note that while modeling temporal effects we should take into account there are transient signals analogous to users and a more long term patterns characterized by movies.

Chapter 2

Collaborative Filtering

In this chapter we explain Collaborative Filtering in more detail, along with giving details of the NETFLIX prize and the lead up to present work. We explain about the data and the approach to the solution of the problem.

2.1 Recommender Systems

Internet technologies have greatly become a part of our lives, there are many occasions when we need help in taking certain decisions. Recommendations in general is ubiquitous when there is a need to make choices without sufficient personal experience, we rely on feedback from users on certain products, we make use of recommendation letters for jobs, reviews of movies and books [19]. Consider for a movie streaming company like NETFLIX, where people can watch movies from a huge collection, have been collecting the ratings of the movies from the users. This data can be considered as our *training data*, it is explicitly obtained from the users. Using this training data the company makes predictions of movies the users are likely to prefer watching more than others. This is only one of the different types of recommender systems.

2.1.1 Recommender System Strategies

The strategies for building recommender systems are based on the type of data available and how the data is used. On a broad sense the two main approaches are *content-based filtering*, in which the items are characterized based on certain attributes, and recommendation are made based on filtering similar items. In the second approach, *collaborative filtering*, the ratings previously given by users is used to build a model, which is then used to make predictions. The term Collaborative Filtering was first termed by Tapestry [10], which was one of the first recommender systems. Recommender Systems produce possibly accurate prediction of relationship between elements of different classes based on previous knowledge or the *training data*. This training data could be explicitly obtained in the form of ratings given by users or it could be implicit knowledge(purchase history, browsing

history, search patterns,mouse movements etc), which can be used when explicit knowledge is insufficient [11]. But however in this thesis work we model our system based on explicit knowledge.

Modern recommender systems are based on the collaborative filtering, which again is of two types, the *nearest neighbor approach* and the *latent factor approach*. In the nearest neighbor methods, items or users are given certain attributes which decides the proximity between them. Further this sense of proximity is used predict an item for an user. The second approach is the latent factor approach, which characterizes the users and the items together in building the model. Matrix factorization is used to build these models, likes of SVD is very useful. *Rating matrix* proves to be important, as it charectarizes both items and users as its vectors.

2.1.2   The Problem Formulation

Our problem could be defined as a matrix completion problem, where the matrix is formed by placing the users u along the rows and the movies m along the rows. The entries in this matrix are filled through the *training data*, which is collection of *n quadruples*, with the format (*user(k)*,*movie(k)*,*rating(k)*,*timestamp(k)*), where $k = 1, 2..., n$. The $1 \leq user \leq u$ and $1 \leq movie \leq m$ are the range of the users and the movies, where as the ratings, $1 < rating < 5$. The *Rating matrix* is defined as $R \in \mathbb{R}^{u,m}$,

$$R[i,j] = \begin{cases} rating(k), & i = user(k), j = movie(k). \\ ?, & \text{otherwise.} \end{cases} \qquad (2.1)$$

where '?' are the cases where the *user-movie* relation is unknown. These are the values which needs to be predicted. Once the '?' values are computed, recommendations could be made depending on the required tolerance level, *user-movie* pairs with predicted ratings higer than the tolerant value(threshold) can be selected for making recommendations. Hence this problem can be viewed as a *matrix completion* problem. In this thesis work, we will compute the unknowns for a certain subset called the *test set*.

Let us denote the completed matrix, or rather the reconstructed matrix by $\hat{R}$. We intend to obtain the $\hat{R}$, by reproducing it through a simple product of two matrices with reduced *inner dimension*. Let $P \in \mathbb{R}^{u,f}$ and $Q \in \mathbb{R}^{f,m}$ are the two matrices with reduced inner dimension *f*.

Let us through a very simple example see the principle behind collaborative filtering.

Example 2.1

We show 4 users who have rated 4 movies, we tabulate the preferences, in the form of ratings on the scale of 1 to 5, as shown below. The users are along the rows and items along the columns, rating given by *user i* to *item j* is the real valued entry

(i,j) in the rating matrix.

	Titanic	Braveheart	The Lion King	Dreamcatcher
John	5	5	2	-
Dave	2	-	3	-
Alice	-	5	-	3
Bob	3	-	-	5

From the above, we can consider the rating matrix to be

$$R_{i,j} = \begin{pmatrix} 5 & 5 & 2 & ? \\ 2 & ? & 3 & 5 \\ ? & 5 & ? & 3 \\ 3 & ? & ? & 5 \end{pmatrix}$$

Using matrix factorization, we simply do a SVD on R to get, U and V, which denote the user space and the movie space.

$$U = \begin{pmatrix} 1.12 & 1.49 & 0.48 \\ 1.31 & -0.52 & 0.59 \\ 1.13 & 0.67 & -0.52 \\ 1.39 & 0.05 & 0.45 \end{pmatrix} \qquad V = \begin{pmatrix} 1.12 & 1.49 & 0.48 \\ 1.31 & -0.52 & 0.59 \\ 1.13 & 0.67 & -0.52 \\ 1.39 & 0.05 & 0.45 \end{pmatrix}$$

Once we have factored the rating matrix R, we can get the missing values by approximating the sparse rating matrix into a full one, $\hat{R} = U * V'$

$$\hat{R}_{i,j} = \begin{pmatrix} 4.78 & 4.98 & 1.97 & 3.61 \\ 1.98 & 1.97 & 2.85 & 4.81 \\ 2.75 & 4.71 & 1.40 & 2.94 \\ 2.94 & 3.32 & 2.75 & 4.79 \end{pmatrix}$$

With this approximation, we can make predictions of ratings of movies which the users have not seen.

2.2   NETFLIX

Entertainment has been one of the most significant integral part of human society and has evloved along with modern technologies. In todays internet driven society, IPTV(Internet Protocol Television) is gaining popularity in delivering television and cinema content to viewers. IPTV mainly consists of three groups,

1. Live Television - Live telecast or transmission to the viewers with only the transmission delay, example live news show.

2. Time-shifted Television which is also called as catch-up TV, in which the viewers can watch the content which is stored and available at any point of time.

3. Video on Demand - here the viewers are given a wide range of TV shows and movies to choose. They can watch anything they wish, the content is streamed to the viewers computer which the user can pause and watch later. NETFLIX provides its content throught this method.

   This owes to a great extent to the success of NETFLIX, since through on Demand system NETFLIX is able to cater to individual tastes of users. NETFLIX can collect statistics pertaining to individual users' choices and the most important metric is the *ratings*. NETFLIX aims to greatly improve the customer satisfaction and retention by providing a greatly personalised experience to the users. The importance of personalization is such, that NETFLIX ignited the research attempts in the Mathematical and Computer Science soceity to develop methods which could do predictions for movie likeability quotient. Such attempts have certainly been fruitful to NETFLIX as is evident from the fact that as of today 33 million members view over 1 billion hours of TV shows and movies through NETFLIX per month.

2.2.1 NETFLIX Prize

During October 2006, NETFLIX challenged the research community to beat the performance in terms of accuracy of their own recommendation system *Cinematch* by 10%. The NETFLIX prize challenge was provided with over 100 million ratings from around half a million users and 18 thousand movies. This data was collected during the period october 1998 and December 2005. The challenge would take place over a period of three years, and a progress prize of 50 thousand USD would be awarded each year to the team which would produce the best improvements. Finally at the end of three years a Grand Prize of 1 Million USD would be given to the team with the best results in terms of RMSE. Their benchmark system *Cinematch*, based on *Pearson correlation* which is straightforward statistical linear model produces an RMSE of 0.9525 on test data. Hence to qualify to win the Grand Prize which accounts for 10% improvement over *Cinematch*, which corresponds to RMSE of 0.8563 on *quiz set*. On an event of a tie the time of entry would be taken into account, and this is exactly what happened on 26 July 2009, team BellKor's Pragmatic Chaos won the challenge by margin of 20 minutes. Table 2.1 shows the performance of top 12 teams, [2] -

2.2.2 The NETFLIX dataset

NETFLIX had been collecting data since October 1998, as of June 2007 the company had collected over 1.9 billion ratings from more than 11.7 million subscribers over 85 thousand titles ReferencesThe NETFLIX Prize. The company then recieved over 2 million ratings per day. To provide data for the competition the company did two seperate random sampling processes, the first one to obtain th eentire Prize dataset and second for the qualifying and probe set. The condition for selecting the users was that they should have given minimum of 20 ratings.

Basically the company has seggregated the data based on time, the latest data

Rank	Team	RMSE	PercentImprovement
1	$BellKor's Pragmatic Chaos$	0.8567	10.06
2	$The Ensemble$	0.8567	10.06
3	$Grand Prize Team$	0.8582	9.90
4	$Opera Solutions and Vandelay United$	0.8588	9.84
5	$Vandelay Industries!$	0.8591	9.81
6	$Pragmatic Theory$	0.8594	9.77
7	$BellKor in BigChaos$	0.8601	9.70
8	$Dace$	0.8612	9.59
9	$Feeds2$	0.8622	9.48
10	$BigChaos$	0.8623	9.47
11	$Opera Solutions$	0.8623	9.47
12	$BellKor$	0.8624	9.46

Table 2.1. RMSE of Top 12 Teams

TRAINING SET	Probe set
100480507	1408395

Table 2.2. Training Data

Test Set	Quiz Set
1408789	1408342

Table 2.3. Qualifying Data

form the Test Data and the initial data form the training data. The *Training data* includes the *Probe set* which is meant for fine tuning before submissions. The training data is basically a text file consisting of triplets entries like, (UserID, MovieID, Rating). There are 100 million such entries which form the training set. The training set also included the probe set consisting of 1,408,395 triplets, this could be used by the competitors to fine tune their algorithms before submission. NETFLIX also collected the latest ratings of all the users in the training set to form the *Qualifying set*. This set was in the form (UserID,MovieID,DatrofRating), where the ratings were known only to the judjes. The qualifying set was again divided into 2 groups, the *Quiz set* and *Test set*. The quiz set having 1,408,342 ratings was used to update the Leaderboard, the RMSE on this quiz set was released to the competitors. The test set having 1,408,789 ratings was only used by the jury to adjudge the winner. [5]

2.3 Matrix Factorization for Collaborative Filtering

With Collaborative Filtering it involves filtering for information through collaboration among interacting groups, for example users and movies. Matrix factorization happens to be the first choice for collaborative filtering, as matrices can hold high dimensional information in tabular form, and can be factored into products of smaller matrices. *In matrix factorization the users and items are mapped onto a joint latent factor space of reduced dimension f, and the inner product of the user vector with item vector gives the corresponding interaction.* [15]

2.3.1 Basic Matrix Factorization

Matrix factorization is mainly about a more compact representation of the large training data which is obtained by dimensionality reduction. [16]. *We want to quantify the nature or the characteristics of the movies defined by a certain number of aspects (factors), i.e we are trying to generalize the information (independent and unrelated ratings matrix) in a concise and descriptive way.* [9] All the movies can be described by certain features like overall rating, cast, wether its an action flick or romantic drama, how did it fare during the first week of it release etc. Similarly a user can also be described based on the same factors, wether the user is liberal in rating or very critical, does the user have certain favourite cast, if the user prefers action over romance, does the user watch and rate movies on the first week of its release etc. What we have here is a common factor space in which we tend to generalize the relationships between the users and movies, based on these factors. Hence the 8.5 billion ratings can be explained in lot fewer factors. We are generalizing the independent individual unrelated ratings to a more concise and descriptive form. Here what we have done is through meaningful generalities, concised the data, the reverse also holds good, i.e by concising the data or in other words by reducing the dimension, we can generalize the description of the large data. The Singular Value Decomposition exactly helps in achieving this. This is explained crudely with an example,

---
Example 2.2

Let us say we have around ten aspects which are used to describe movies and users. Each movie has ten numerical values, which exemplifies the ten aspects. Similarly the users also have ten such values. We can combine these values to obtaing a single rating value for each user-movie pair, bu multiplying the corresponding terms and summing them up. Let us consider a movie Die Hard whose aspects might be something like, action=1.2, chickflick=-1, and so on. Let us consider a user Joe whose aspects are, action=3, and chickflick=-1. So the overall rating for the movie Die Hard given by Joe is, $3 * 1.2 + -1 * -1 + .. = (3.6 + 1 + ..)$. It clearly shows that Joe would evetually like the movie Die Hard, which is evident from the high rating value obtained.

The model in its most basic form is built around the following $R = P * Q'$ where $P \in \mathbb{R}^{u,f}$ and $Q \in \mathbb{R}^{i,f}$. The number f is decided by trial and error, higher the value better the predictions, however beyond certain value it will not contribute to the accuracy significantly. This value decides the number of *factors* which will be used to characterize users and movies. This is analogous to dimension reduction.

The individual vector in P, $p_u \in \mathbb{R}^f$ represents an user, it gives a measure of the qualities represented by the f factors. Similarly, every item is represented by a $q_i \in \mathbb{R}^f$ which says about the f qualities pertaining to the items, i.e movies. We cannot for sure say what are the qualities which these factors represent. The task of predicting a rating for a user u gives for a movie i, can be computed as an inner product,

$$\hat{r}_{ui} = q_i^T p_u.$$

This dot product gives the iteraction between the item i and the user u. The challenge is in the mapping of all the users and items to a joint latent factor space of dimensionality f [15].

Singular Value Decomposition (SVD), would be first choice for such a factorization, as it helps a great deal in identifying the the best top-most factors, i.e by selecting the left and right singular vectors corresponding to top n singular values.

2.3.2 Dimension Reduction

Matrix factorization and dimension-reduction go hand-in-hand, the dimension of our rating matrix is very high and along with factorizing the matrix it is important to reduce the dimensions of the factored matrices. It is the inherent property of SVD that we can decide the rank or the number of *factors* we need in order to get the best approximation of our User-Movie rating model. By dimension reduction we get to express the information which was initially presented in a matrix of dimension 480189x17770 having around 85 billion ratings in much smaller space with only k factors, where k is the reduced dimension. While a reduction in dimension, also aids in removal of noise, however too low dimension can result in removal of useful information as well. Hence it is important to decide on an optimal dimension. Through an example below we can further justify the need for dimension reduction,

Example 2.3

Consider movies like "Harry Potter", "Chronicles of Narnia", "Lord of the Rings" etc, there are very large number of movies similar to this. By saying similar what we mean is that these movies all have certain common factors, like they all are

$Model No.$	$Model$
1	Naive Models based on user/movie means
2	Bias-Model
3	Temporal model - SVD
4	Temporal model - ALS

Table 2.4. Different models

fantasy movies, they all are novel based movies. Hence instead of retaining all these movies, we can replace them with certain latent factors, which are much fewer than the number of items or movies, and hence the dimension is reduced.

2.3.3 Different Models

Collaborative Filtering based Recommender Systems are built using either Neighbourhood-methods or model-based methods [12]. Our approach involves the model-based method, which has proved to be one of the most effective approach as is evident from the NETFLIX Challenge and large number of research articles on the same topic. To have a good Recommender System, it is important to achieve good accuracy and also to keep the model simple. Keeping this in mind, we try to build a model which accounts for significant predictor variables, like the temporal dynamics, user-bias, movie-bias etc. However we initially analyze these individual predictors and after careful analyses, we blend them together in the final model to achieve the best RMSE. The main principle behind our model of laten factor modelling, supplemented with the removal of biases pertaining to users, movies and temporal variations. Below table shows the various models we use,

Data Representation

Before beginning discussion on the models, we show a detailed representation of the training and the test data which are used in this work, which are in the form of matrices, i.e, the rating matrix R, the date matrix D, these comprise the training data. The test data is in the form of three matrices, probe set P_{probe}, test set Q_{test}, quiz set Q_{quiz} and finally Q_{all} which consists of the entire test data. Each of the above matrices except D are in the form,
$\mathcal{R} = \{r_{ij} \exists \text{ in } R \mid \text{user } i \in \mathcal{U} \text{ have rated movie } j \in \mathcal{M}\}$,
where r_{ij} is the rating for the corresponding user-movie pair and is also the $(i,j)^{th}$ entry in their respective matrices. The matrix D however has the same structure of R, but with the date of rating instead of the rating itself.

Basic Naive model

We start by doing some rating predictions based on the average of all the ratings in the R matrix, which is 3.5769. We simply use this value for any unknown

user-movie combination. Moving ahead we could use the *user-mean*, to calculate the unknown rating, for example if we need a prediction for user u, for any of the movies we simply use the average of all the ratings given by user u. Similarly to predict the unknown rating for some movie i, we could simply use the average rating of that movie over all the known ratings given to that movie in the matrix R.

Bias model

Moving ahead we try out new models which takes care of the inherent biases of various kinds in the data. Bias model removes the anomalies, if any in the data. Biases in this cintext can be considered to be variations in the ratings which are caused by certain effects associated with the users or movies, independetly of the interaction between the two groups. The interaction is captured in the factorization part and the inner products of the interacting vectors. The main purpose of doing this is, by separating the interaction and the biases will help us in subjecting only the interaction portion of the data which is more critical [13]. The *Baselinepredictors* capture these biases which do not involve user-movie interaction. While factorization these biases are removed only to be added after factorization, i.e, post-processing.

Temporal Bias + SVD model

The main objective in this thesis is to capture the temporal dynamics within the data and include them into the *baselinepredictors* through two major temporal effects, each involving users and movies. However in this thesis the temporal effects related only to movies is considered. We have considered the movie-bias to be a function of time, which can capture the changes in popularity of movies in course of time, owing to number of external triggering factors like the actor in the movie winning a major award. Also the users tend to drift with their rating styles with time, for example a liberal user might for some reason become a more critical in rating movies.
This model uses the good old Singular Value Decomposition for the factorizing the rating matrix. But however it is evident that owing to the sparsity in the data SVD proves to be not very effective. To avoid this imputation can be employed but this can be very expensive in terms of memory.

Temporal Bias + ALS model

Owing to the disadvantage of the SVD as will be explained in future chapters, we employ an alternative method in Alternating Least Squares. It is an iterative method with initial guessed values for M and alternatively solving for U and M until convergence. This method is computationally demanding but can be parallelized easily. If R is the rating matrix, we formulate a low-rank matrix approximation problem as shown below,

$$(U, M) = arg \min_{U,M} \mathcal{L}^{emp}(R, U, M) \qquad (2.2)$$

where U and $M \in \mathbb{R}$ and have n_f columns. The symbolic representation of the problem is shown below,

The x^s denote the known ratings in the matrix R and $\in I$, where I is the known ratings set and n is the number of known ratings in total. In general the problem is of the type,

$$\|Ax - b\| \tag{2.3}$$

$$\|R - UM^T\| \tag{2.4}$$

where we are trying to minimize the above in Frobenius norm, which is matrix equivalent of 2-norm. We wish to solve the above problem in a least squares sense, but since it is not in the form of ordinary least squares(OLS). We try to convert the type shown in Equation 2.4 to the type shown in Equation 2.3.
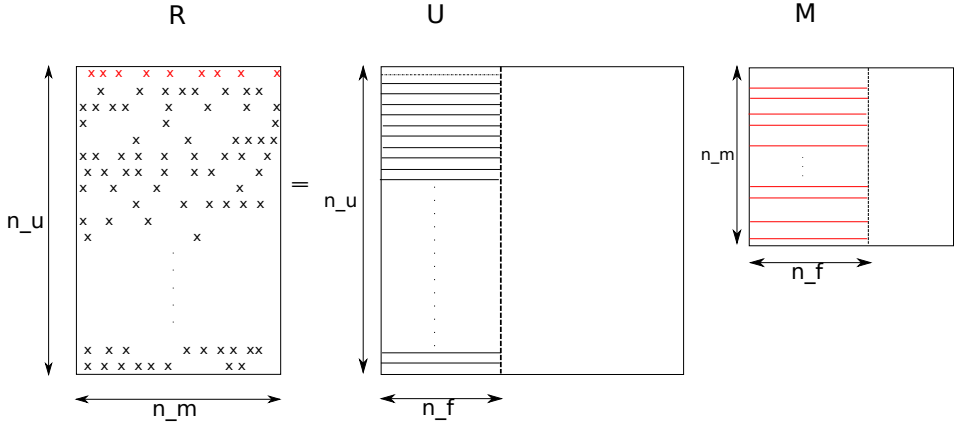

Figure 2.1. ALS Decomposition

Let us consider the first row of R, which will have all the ratings given by the fist user. Let us assume the user has seen movies 1, 213, 315, 1744, 2133, 7044, 9012, 12344 and 16890 so in a total of 9 movies. Forming a vector of the known ratings for the first user we have $\begin{bmatrix} 1 & 213 & 315 & 1744 & 2133 & 7044 & 9012 & 12344 & 16890 \end{bmatrix}$, let this be denoted by b. We initialize the M matrix in a particular way, and as shown in the Figure 2.1 it has n_m rows and n_f columns. The matrix U has n_u rows and n_f columns. For the case of the first user we can form a sub-matrix of M, denote this by M_u. This matrix M_u will have all the rows from M corresponding to the movies seen by the user u, the first user in this case. Let u be the row vector from U corresponding to the considered user, .i.e, first user. Now we have three components $R(1,:)$, the row vector having the nine ratings of the nine movies seen by user 1, u_1 the unknown first row of U matrix, M_u, a submatrix of M consisting of the rows corresponding to the movies rated by the user 1. Figure 2.2 shows the symbolic representation of this single ordinary least squares problem formulation, $\|M_u u - R(1,:)\|$. This is done for all the users to obtain the U matrix. Next this U matrix is used to estimate the new M matrix. The U and M are estimated alternatively until some convergence criteria is satisfied.

R(1,:)   u   M

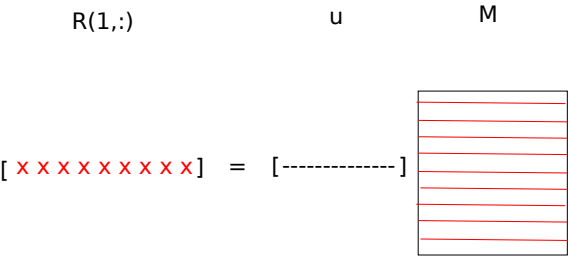[x x x x x x x x x] = [--------------]

Figure 2.2. OLS

Chapter 3

Temporal Dynamics

In this chapter we explain all the aspects related to Temporal Dynamics in the NET-FLIX data. We also show how to build temporal bias based model.

3.1 Temporal Dynamics in NETFLIX data

The main goal in this work is to integrate temporal information into the Recommender System model we are building based on collaborative filtering. It is very important to consider temporal dynamics since the vast amounts of data we are flooded with is dynamic in nature, and most of the information retrieval tools today consider only the snapshot of this data, thereby loosing one very potential dimension. In this chapter we present in detail the significance of temporal dynamics in general and with respect to NETFLIX and also how to include temporal dynamics into the model.

Since we are trying to improve the prediction rate by including temporal effects, it is very critical to identify and uncover all possible temporal effects within the NETFLIX data. This section summarizes and reports the temporal analysis of the NETFLIX data.

3.2 Tracking Drifting Customer Preferences

The phenomena of $Concept drift$ is evident in the case of NETFLIX system, as we can see that the user preferences of movies keep changing over time due to various reasons. Concept drift means when the statistical properties of the target variables change with time, like how user preferences change with new release of new movies. In our case where we are trying to collaborate using interconnected preferences among users, it is a challenge since each of the interacting users could be drifting in different ways and also they are in different time instances. There are mainly three kinds of approaches to include time apect, *instance selection*, in which only the recent instances are considered, discarding off other instances.

All instances within a specific time period is given equal opportunities discarding instances ouside this period. This obviously leads to lose of information. In 2005, a time-weighted CF method was proposed by Ding and Li in [6], in which depending on how recent the ratings are they are weighted accordingly, these models do include time but only as weighted schemes, this is the second approach called *instance weighting*. The third approach is the *ensemble learning*, in which predictions from number of models are combined to give a final prediction. But in this approach, if each of the individual predictors capture local behaviour, the global patterns are overlooked. A more complex approach was proposed by Koren et al, in [13], in which the various complex time parameters are incorporated into the model. We have built our model based on the following guidelines,

- We try to model for the entire time period, instead of any particular instances or particular time periods.

- We try to capture the multiple drifts based on the user, both sudden changes and gradual changes.

- All the various drifts will be combined into a single framework.

Example 3.1

As an example consider two users Joe(21 years old) and John(41 years old) rating a common movie Godfather(1972). During November 2008, Godfather was voted No. 1 in Empire magazine's list of The 500 Greatest Movies of All Time. Joe after seeing this watched the movie and gave his rating. On the other hand John had rated the movie 20 years back. This makes them both roughly the same age when they have rated the movie, however in a different time instance. It is clear that Joes rating could have had certain bias caused by *classical* nature of the movie.

3.3 Temporal Dynamics in NETFLIX Data

Here we would show through plots the dynamic nature inherent in the data with respect to time.

3.4 Temporal-Factor Model

As we had seen in the previous chapter, that all the non user-item interaction effects are encapsulated in the *Baseline predictors*, we will show here who we intend to include temporal baseline predictors. Here we explain the theoretical analyses and findings related to various time changing aspects of users and of movies. It is clear that the time dynamics of movies will be more gradual than that of the
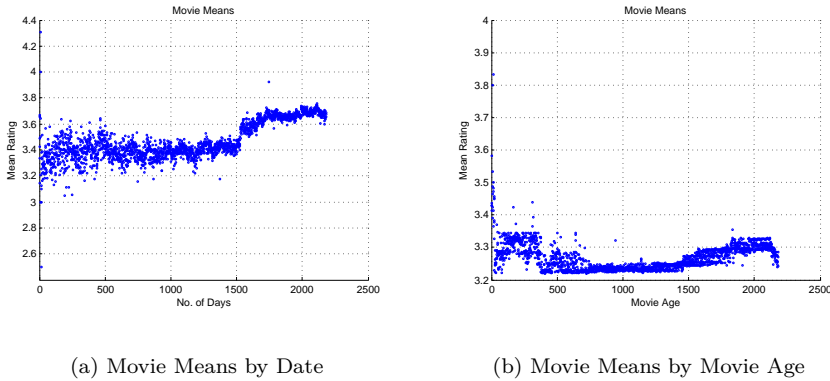
(a) Movie Means by Date   (b) Movie Means by Movie Age

Figure 3.1. Movie Means vs Time

users. The movies are analysed based on a large number of users who have rated
the single movie, but however as an individual user the temporal analyses is done
on large number of unrelated factors, hence the user behaviour seems more com-
plex and less uniform compared to the movies.

The main challenge is to parameterize the user-drift and the movie-drift as appro-
priately as possible. The first and foremost consideration is how transitory are the
user-drift and the movie-drift. As we will expalin in detail the movie-drift seems to
be more uniform and less transient than the user-drift, hence it is wise to consider
finer time resolution to model user-drift and coarser resolution for movies. The
final model should however account for both temporal effects spanning on a longer
duration and the more transitory changes. For example, movies dont change on a
daily basis, some of the factors which could affect the movies dont occur often like
appearance of an actor in another movie, or the movie winning an award or the
movie recieving appreciation from a critique. On the contrary consider an user,
who usually rates relative to other ratings, so his rating scale could vary everytime
he has seen on influencing movie prior to the current movie of interest. A user
who usually rate 4 for average movies, could be influenced to rate the same kind
of average movies by 3.

We begun by experimenting for different time periods called *bins* in order to
catch the movie-drifts. The whole time period spans over 300 weeks, and after
several rounds of testing a 10 week period was chosen as the optimal bin length.
Hence a total of 30 bins suffice to span the entire data. With each bin we as-
sociate it with a bias(transient part), which is simply the difference between the
overall movie mean and the mean of the movie for that bin period. Interpolation
is done for bins where there are no ratings. Each rating instance of a movie in the
training data is associated with a bin value between 1 and 30. Hence the movie
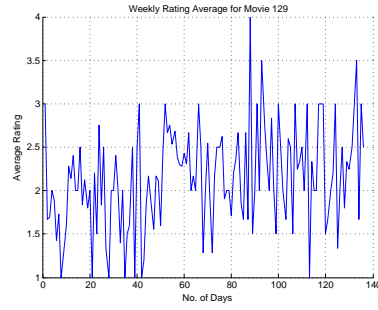bias would have two components a stationary part, b_i and a transient part $b_{i,Bin(t)}$
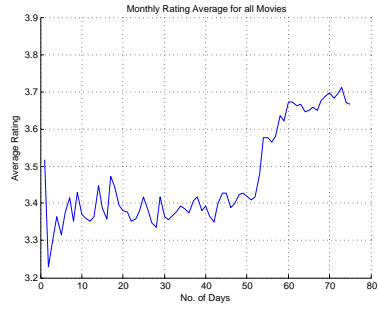
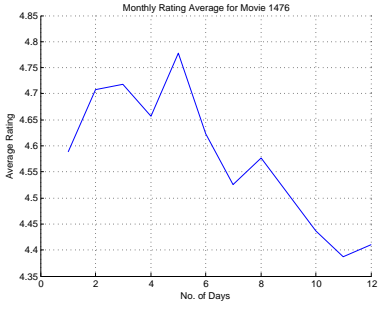(a) Monthly Average of All movies
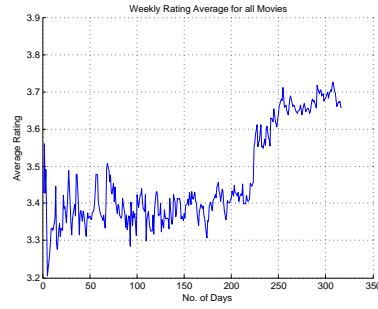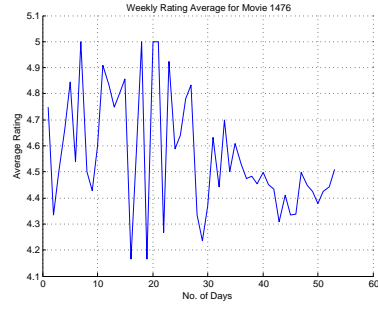
(b) Weekly Average of all movies
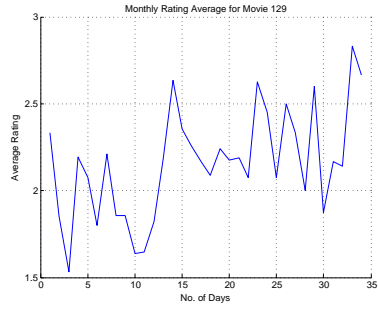
(c) Monthly Average of movie 129

(d) Weekly Average of movie 129

(e) Monthly Average of movie 1476

(f) Weekly Average of movie 1476

Figure 3.2. Monthly and Weekly Averages

References[7].

$$b_i(t) = b_i + b_{i,Bin(t)} \tag{3.1}$$

We have in this thesis work considered only movie related temporal dynamics, however we would comment on a possible approach to capture temporal dynamics related to the more complicated user bias. In case users we need to parameterize for both long term and short term changes. A simple linear function could be used to capture the gradual user-bias drift. Each user will have an average rating date, and for every rating instances of this user the deviation could be a function of the difference of the dates. Regarding sudden drifts, we need to capture the drift on daily basis. The minimum step length could be larger than a day also. In the NETFLIX data the user on an average rates on 40 different days, hence we need 40 different paramaters for every user on an average. These two factors could model the temporal dynamics of the users.

Chapter 4

Mathematical Background

In this chapter we provide the mathematical proofs which form the basis of the prediction models. The main concept of matrix factorization is explained mathematically. Singular value decomposition is explained in detail, and some issues regarding the disadvantages in the present scenario is explained. An substitute method Alternate Least Squares is explained. And lastly low rank approximation is explained.

4.1 Least Squares and Orthogonality

4.1.1 Linear Systems

The nature of the problem and the data we are dealing with is linear in nature, and hence we should start by looking into the basic principles involved in solutions of linear systems. Almost all the complex linear problems when broken down to the most granular level, can be viewed as,

$$Ax = b, \tag{4.1}$$

where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. It is applied to deal with real life problems, by considering x to be vector of causes and b to be the vector of effects of a system which is characterized by matrix A. So when we know the causal vector x we can find the effect b, by $b = Ax$. Or in most data modeling problems, we know the effects(from the *training data*), b, we need to find the causal vector x.

If $m = n$, A happens to be square and it is expected to be nonsingluar in order to have a unique non-trivial solution. Here the vector b can be represented as a linear combination of the columns of A. This simply means that $b \in span(A)$, where $span(A) = Ax : x \in \mathbb{R}^n$

However not all problems in real life is as simple as this, we have to deal with rectangular sytems, i.e., $m \neq n$. Well on most scenarios when this happens, we have $m \geq n$, and we have different iterative approaches to its solution.

4.1.2   Vector and Matrix Norms

Since we will be approximating the unknown vectors and eventually matrices, we need to have way to measure their 'sizes'. Hence we need to apply the scalar concepts of, magnitude, absolute value, or modulus to vectors and matrices.

We can measure the 'size' of vectors by,

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|, 1 - norm,$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}, 2 - norm,$$

$$\|x\|_\infty = \sum_{i=1}^{n} |x_i|, max - norm,$$

In a generalized form, the *p-norm* is defined as,

$$\|x\|_2 = \sqrt[p]{\sum_{i=1}^{n} x_i^p}$$

The matrix norms for matrix $A \in \mathbb{R}^{m \times n}$ as computed as follows,

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

$$\|A\|_2 = (\max_{1 \leq j \leq n} \lambda_i(A^T A))^{1/2}$$

Our main goal in this thesis work is to approximate a matrix by predicting the missing values. In order to do this we minimize the residual, r, between the known rating matrix $R \in \mathbb{R}^{m \times n}$ and the approximating matrix $\hat{R} \in \mathbb{R}^{m \times n}$. While finding the residual, we treat the two matrices, R and $\hat{R}$ as points in $\mathbb{R}^{mn \times mn}$, and we use a special *Frobenius norm*, to calculate the distance between them,

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}.$$

4.1.3   Bases and Rank of Matrix

Suppose we are dealing with a real space, $\mathbb{R}^m$, it is handy to have a fixed number of vectors, making use of which we can span the entire space under consideration. This collection of fixed number of vectors is called *basis* in $\mathbb{R}^m$.

Let us consider a set of vectors $(v)_{j=1}^{n}$ in $\mathbb{R}^m$, $m \geq n$,

$$span(v_1, v_2, ..., v_n) = \left\{ y | y = \sum_{j=1}^{n} \alpha_j v_j \right\}$$

for arbitrary coefficients α_j. The vectors $(v)_{j=1}^n$ are *linearly independent* when

$$\sum_{j=1}^{n} = 0 \; if f \alpha_j = 0 \, for \, j = 1, 2, ...n.$$

The maximum number of linearly independent vectors(column or row), is called the *rank* of the matrix [7]

4.1.4 Least Squares

In almost all problems in which numerical techniques are used to find solutions, we have to deal with *overdetermined* system,

$$Ax \cong b,$$

where $A \in \mathbb{R}^{m \times n}$, with $m > n$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. In this case it would not be possible to represent m-dimensional b, as linear combinations of only n column vectors of A. It so happens that the points on the LHS and RHS, are not in a common space. Hence we would look to minimize some norm of the *residual*, $r = b - Ax$. The solution obtained here is in the *least squares* sense.

   Let us classify the problems, as type-I, in which we have to deal with exactly or very linearly dependent vectors, and secondly, type-II, in which the system consists of almost linearly dependent vectors as well. Direct linear solvers like Gaussian elimination can be applied to type-I problems only. To deal with type-II, which usually has noise in it, we need to filter out the bad basis vectors, which are represented by the almost linearly dependent vectors. In order to quantize the difference between the *very* and *almost* linear dependence, we make use of orthogonality. The good vectors, i.e., the very linearly dependent vectors are close to orthogonal.

4.1.5 Orthogonality

Why is *orthogonality* important to us? Our basic approach in building the prediction model, is to factorize the rating matrix $R \in \mathbb{R}^{m \times n}$ with $m > n$, into $P \in \mathbb{R}^{m \times k}$ and $Q \in \mathbb{R}^{n \times k}$, where generally $k << n < m$. By reducing the dimensions, we are removing the bad basis vectors, i.e., vectors carrying no significant useful information. It is strongly desired that the remaining good basis vectors be orthogonal to each other.

   Two nonzero vectors x and y are orthogonal to each other, if $x^T y = 0$, which is the same as $cos\theta = 0$, where θ is the angle between x and y.

Proposition 4.1 *Let q_j, j=1,2,...,n, be orthogonal, i.e., $q_i^T q_j = 0$, ij. Then the vectors q_j are all linearly independent.*

   We will now see how we can decompose the matrix to more compact forms, using orthogonal transformations. QR decomposition is one stable technique to achieve this. QR decomposition is not very useful for our problem due to various

reasons, which will me mentioned later, it helps in understanding the concepts and gradually take us to the more complex methods like SVD, which will be introduced later. The matrix $A \in \mathbb{R}^{m \times n}$, which is usually the data matrix, can be factorized as shown below,

$$A = Q \left(\begin{array}{c} R \\ 0 \end{array} \right),$$

where $Q \in \mathbb{R}^{m \times m}$ is orthogonal and $R \in \mathbb{R}^{m \times n}$ is upper triangular.

The main reason to factorize the data matrix are,

1. Compact representation

2. Representation in terms of *basis* in column-space, row-space or both column and row space.

Below Figure 4.1, show how QR factorization achieves the above goals, i.e., firstly compact representation, which is achieved by reducing the *rank= k*. It can be shown that by optimally choosing the rank k, we get better performance. This value of k, can be chosen by trial and error methods. Secondly, QR factorization helps us in representing the matrix in terms of the basis vectors in the column space or in the row space. In the figure below, we can see that the matrix Q, forms the orthonormal basis for the column space of A. This means that, the factors in the column space if A defines or characterizes the system. This means that the known information is used only from only one dimension. So this technique is more suitable for solving linear systems in least squares sense, as in Equation 4.1, where only the column-space is sufficient, as the known vector b lies in the column space. We shall consider an example application, to study the concepts discussed. This example is obtained from [7].

4.2 Introduction to Matrix Factorization

Since this chapter is dedicated to the mathematical principles that drives the prediction models, we can begin with a direct statement, the data matrix is factorized into simpler matrices which are constrained on its dimensionality. Different types of constraints can be imposed on the factorization, but for the particular machine learning task of collaborative filtering we have only used the low rank approximation. Collaborative filtering can be considered as *matrix completion* problem, where we are trying to fill in the missing entries of the sparsely filled rating matrix $R \in \mathbb{R}^{u,i}$. We complete the rating matrix by approximating it to a full matrix $\hat{R} = PQ'$, where $P \in \mathbb{R}^{u,k}$ and $Q \in \mathbb{R}^{i,k}$.

P and Q are unconstranied *factor matrices* and we will show that if $\hat{R}$ is the best approximation of the original rating matrix, then the *rank* of $\hat{R}$ is at most k.

Now that we have had a fair idea of how to approach the problem, the next important concern is to decide in what sense do we want to approximate the data matrix, or how do we measure the discrepency between the original matrix and the approximated matrix. The simplest way would be to measure the *Frobenius*
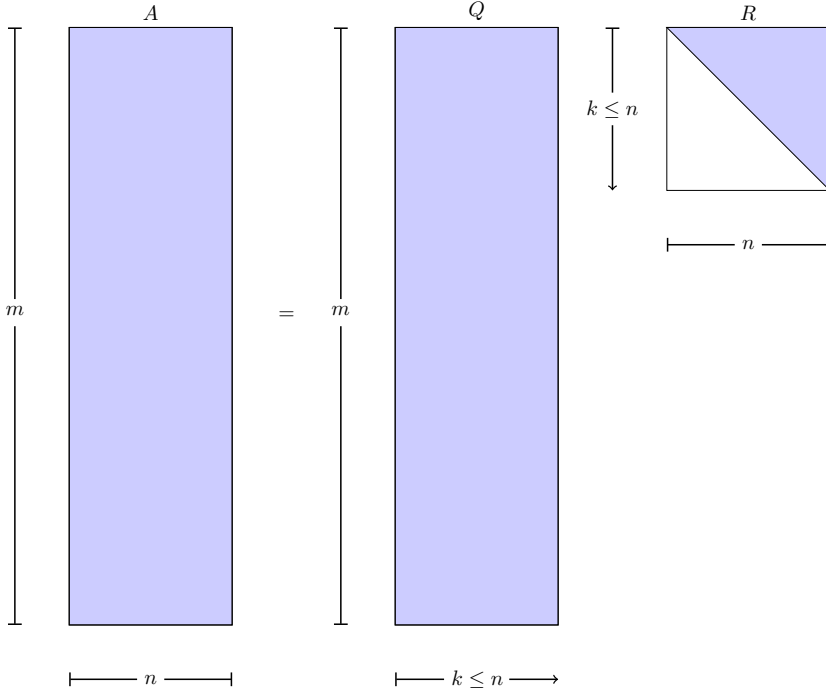
Figure 4.1. QR Factorization with reduced rank, k

distance between the two,

$$\|R - \hat{R}\|_F^2 = \sum_{ui}(R_{ui} - \hat{R}_{ui})^2$$

But we use the Root Mean Squared Error(RMSE) to measure the accuracy.

There might also be a necessity in applicatons like clustering to impose a constraint on the factored matrices other than only on rank of approximating matrix. Since clustering is based on some kind of distance measure and distance can never be negative, it is obvious that we need to impose a non-negative constraint on the factored matrices. In such cases we cannot rely on SVD as it can generate negative elements. Instead we might want to compute the rank-k approximation in the following way [7],

$$A \approx WH, W, H \geq 0$$

4.3   Matrix approximation using SVD

Theorem 4.2 (SVD) *Any $m \times n$ matrix A with rank k, with $m \geq n$, can be factorized*

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T, \tag{4.2}$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix, i.e., $\Sigma = diag(\sigma_1, \sigma_2, ... \sigma_n)$, with $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$. Here $\sigma_1, ..., \sigma_n$ are called singular values of A. The column vectors of U and V are called the right and left singular vectors of A, respectivley.

The matrix A can be constructed from the singular values as shown below,

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T \tag{4.3}$$

The matrix U can be written as $U = [U_1 U_2]$, where $U_1 \in \mathbb{R}^{m \times n}$. We can obtain the matrix A by the *outer product form* in a thin version as shown below,

$$A = U_1 \Sigma V^T = \begin{pmatrix} u_1 & u_2 & \cdots & u_n \end{pmatrix} \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{pmatrix}$$

Now moving ahead with matrix approximation, let us consider our original matrix A, as sum of a low-rank matrix and a small noise, $A_0 + N$. Since it is an approximation problem we will try to minimise the noise N, by estimating the *correct rank* of A_0, this rank is known as *numerical rank*. [7]. Now the approximation looks like this,

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T \approx \sum_{i=1}^{k} \sigma_k u_k v_k^T$$

Theorem 4.3 (Eckart-Young theorem) *For a matrix $A \in \mathbb{R}^{m \times n}$, with $rank(A) = r > k$. The Frobenius norm of the matrix approximation problem*

$$\min_{rank(\hat{A})=k} \|A - \hat{A}\| \tag{4.4}$$

which has the solution of

$$\hat{A} = U_k \Sigma_k V_k^T, \tag{4.5}$$

where $U_k = \begin{pmatrix} u_1 & u_2 & \cdots & u_k \end{pmatrix}$, $V_k = \begin{pmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{pmatrix}$ and $\Sigma_k = diag(\sigma_1, \sigma_2, ..., \sigma_k)$. Equation 4.3 has the minimum solution of

$$\|A - \hat{A}\|_F = \left(\sum_{i=k+1}^{p} \sigma_i^2 \right)^{1/2}, \tag{4.6}$$

where $p = min(m, n)$.

4.4 Matrix approximation using Alternating Least Squares

4.4.1 Disadvantage of SVD

In this thesis work we have done the implementation using MATLAB, and we use the function $[U, S, V] = svds(A, k)$ to obtain the singular value decomposition of a sparse matrix A of rank k. However elegant is the method of SVD, it unfortunately fails to produce good predictions. The SVD approximates a rating matrix by minimizing the *Frobenius norm* of $|R - \hat{R}|$. This is equvivalent to minimizing the RMSE between individual elements in the rating matrix. Since the majority of the elements are *unknowns*, and it is our belief that MATLAB while calulating SVD considers these *unknowns* as zeros, whereas these are not zeros and hence this approach is flawed. In the Appendix A, we give a detailed explanation as to why SVD is not appropriate to find the low-rank matrix approximation in our case, where our data matrix is very sparse(1%-dense).

4.4.2 Problem formulation w.r.t ALS

This approach is based on the article [20]. Let $U = [\mathbf{u_i}]$ be the user feature matrix where $\mathbf{u_i} \subseteq \mathbb{R}^{n_f}$ and $i = 1, 2, ..., n_u$, and let $M = \mathbf{m_j}$ be the item or movie feature matrix, where $\mathbf{m_j} \subseteq \mathbb{R}^{n_f}$ and $j = 1, 2, ..., n_m$. Here n_f is the number of factors, i.e., the reduced dimension or the lower rank, which is determined by cross validation. The predictions can be calculated for any user-movie combination (i, j), as $r_{ij} = \mathbf{u_i} \cdot \mathbf{m_j}, \forall i, j$. Here we minimize the loss function of U and M as the condition in the iterative process of obtaining these matrices. Let us start by considering the loss due to a single prediction in terms of sqaured error:

$$\mathcal{L}^2(r, \mathbf{u}, \mathbf{m}) = (r - < \mathbf{u}, \mathbf{m} >)^2. \tag{4.7}$$

Based on the above equation generalizing it for the whole data set, the *empirical total loss* as:

$$\mathcal{L}^{emp}(R, U, M) = \frac{1}{n} \sum_{(i,j) \in I} \mathcal{L}^2(r_{ij}, \mathbf{u_i}, \mathbf{m_j}), \tag{4.8}$$

where I is the known ratings dataset having n ratings.

Based on the above, we can formulate our low rank matrix approximation problem as

$$(U, M) = arg \min_{(U,M)} \mathcal{L}^{emp}(R, U, M). \tag{4.9}$$

Here the number of elements or free parameters we need to determine is $(n_u + n_m)n_f$, but in our initial known data matrix, .i.e, R we only have 1% of $n_u n_m$ elements. While solving Eq. (4.8) with a sparse R matrix leads to overfitting. To avoid this we use Tikhonov regularization, our emperical loss term get another term as shown below:

$$\mathcal{L}_\lambda^{reg} = \mathcal{L}^{emp}(R, U, M) + \lambda(\|U\Gamma_U\|^2 + \|M\Gamma_M\|^2), \tag{4.10}$$

where Γ_U and Γ_M are Tikhonov matrices. In the next section we will discuss in detail about how to form these Tikhonov matrices.

4.4.3   Regularized ALS

According to the article [20], it is best to use weighted-λ-regularization, which is as shown below:

$$f(U, M) = \sum_{(i,j) \in I} (r_{ij} - \mathbf{u}_i^T \mathbf{m}_j)^2 + \lambda(\sum_i n_{u_i} \|\mathbf{u}_i\|^2 + \sum_j n_{m_j} \|\mathbf{m}_j\|^2), \tag{4.11}$$

The above equation can is somewhat an vectorized version of Eq. (4.9), where the Tikhonov matrices are replaced by n_{u_i} and n_{m_j}, which are nothing but the number of ratings of user i and movie j. Below equations show how these terms are related to Tikhonov matrices:

$$\Gamma_U = diag(n_{u_i})\Gamma_M = diag(n_{m_j})$$

Now we will discuss how to solve for matrices U and M, this is the mathematical explanation of what was explained under Temporal Bias + ALS model in Sec. (2.3.3). The matrix U is approximated column wise, i.e., u_i, is determined by solving the regularized least squares problem, using the known ratings vector of user i and the feature vectors m_j, which are the columns of M matrix corresponding to the movies seen by the user i. Let us represent Eq. (4.10), is a simple differential equation style as follows:

$$f(u, m) = (r - um)^2 + \lambda(nu^2 + nm^2) \tag{4.12}$$

to minimize this function we equate the first partial differential w.r.t u of the function f to 0.

$$\frac{1}{2}\frac{\partial f}{\partial u_{ki}} = 0, \qquad\qquad \forall i,k \qquad (4.13)$$

$$\implies \sum_{j\in I_i}(\mathbf{u}_i^T\mathbf{m}_j - r_{ij})m_{kj} + n_{u_i}u_{ki} = 0, \qquad\qquad \forall i,k \qquad (4.14)$$

$$\implies \sum_{j\in I_i}m_{k}j\mathbf{m}_j^T\mathbf{u}_i + \lambda n_{u_i}u_ki = \sum_{j\in I_i}m_{kj}r_{ij}, \qquad\qquad \forall i,k \qquad (4.15)$$

$$\implies (M_{I_i}M_{I_i^T} + \lambda n_{u_i}E)\mathbf{u}_i = M_{I_i}R^T(i, I_i), \qquad\qquad \forall i,k \qquad (4.16)$$

$$\implies \mathbf{u}_i = A_i^{-1}V_i, \qquad\qquad \forall i \qquad (4.17)$$

$$(4.18)$$

where E is $n_f \times n_f$ identity matrix, $V_i = M_{I_i}R^T(i, I_i)$ and $A_i = M_{I_i}M_{I_i}^T + \lambda n_{u_i}E$.

Now in the same way by starting with a known U matrix, the movie feature matrix M is approximated by computing the individual column vectors of M as shown below:

$$\mathbf{m}_j = A_j^{-1}V_j, \qquad\qquad \forall j,$$

where $A_j = U_{I_i}U_{I_i}^T + \lambda n_{m_j}E$ and $V_j = U_{I_i}R(I_j, j)$. U_{I_i} is the sub-matrix of U, where only $i \in I_j$ are selected. $R(I_j, j)$ is the column vector which is obtained from the rating matrix R, by selecting all the elements from column j and rows corresponding to $i \in I_j$

4.5   Evaluation Metrics

With the advent of the NETFLIX Competition, *accuracy* has become one of the most important defining properties of Recommender systems. Hence it is worth-while to discuss about different methods of measuring accuracy. The choice of metrics for accuracy might depend on the kind of systems under study, like if it is *rating prediction system* or *decision support system*. For the former we can use statistical accuracy measures where we compare the predicted value with known actual values, for example the *MAE(Mean Absolute Error)*, *RMSE*, *Correlation* etc. In the case of decision support systems, where it is required to produce a list of high-relevance items, like in the *top-N* recommendation systems, the metrics used to measure accuracy are *precision*, *recall*, or *F-measure* etc.

4.5.1 Mean Average Error

Let us consider, $\mathcal{T} = \{r_{ij} \exists \text{ in } Q_{\text{test}} \mid \text{user } i \in \mathcal{U} \text{ have rated movie } j \in \mathcal{M}\}$, to be the test set. Then the MAE can be defined as,

$$MAE = \sum_{r_{i,j} \in \mathcal{T}} \frac{|\hat{r}_{ij} - r_{ij}|}{|T|} \tag{4.19}$$

where $\hat{r}_{ij}$ is the predicted rating and r_{ij} is the actual rating, and $|T|$ is the size of the test set. The goal is to minimize this value to get better accuracy.

4.5.2 RMSE

For the same test set as shown above, the RMSE is defined as,

$$RMSE = \left(\frac{\sum_{r_{i,j} \in \mathcal{T}} (\hat{r}_{ij} - r_{ij})^2}{|T|} \right)^{1/2} \tag{4.20}$$

4.5.3 Correlation

If we consider the predicted and the known values to be vectors, then the correlation gives the proximity of the two vectors. *Pearson Correlation Coefficient* is an example, which is defined by

$$\rho_{\hat{r},r} = \frac{cov(\hat{r}, r)}{\sigma_{\hat{r}} \sigma_r} \tag{4.21}$$

Higher the value better the accuracy in this case. Geometrically this can be viewed as the *cosine* of the angle between the two vectors.

4.5.4 Precision and Recall

In the general *Information retireval* point of view, the precision is defined as,

$$P = \frac{D_r}{D_t} \tag{4.22}$$

where D_r is the number of relevant documents retrieved and D_t is the total number of documents retrieved [7]. But in the Recommender system point of view the documents are replaced by items, and the user interest according to the user profile is used instead of the explicit query.

Recall is defined as,

$$R = \frac{D_r}{N_r} \tag{4.23}$$

where N_r is the total number of relevant documents in the database, which in the case of recommender systems would be the total number of items in the whole data.

4.5.5   F-Measure

This is a way to combine the precision and recall into one metric, a simple F-1 measure is the harmonic mean of precision and recall.

Chapter 5

Results

In this chapter we present the results of the epxeriments conducted, reasoning the motivation towards the built models. We also talk about aspects which were considered but not implemented.

5.1   Overview of Different Models

The principles behind designing the model are firstly, to accurately account for biases of various types and secondly, formulation of an efficient low-rank matrix approximation method. The first principle involves identifying different biases, like, the user-bias, movie-bias and temporal biases w.r.t users and movies. These biases are the non-interacting parts between the users and movies, and they are encapsulated into the baseline predictor. These values are initially subtracted from all the known ratings. They will be integrated back into the factor model for the new predicted ratings. We have experimented with three types of biasing techniques, first without any bias, second, with only static bias, and third, with temporal movie biases. Moving ahead to the factorization itself, we start with the SVD technique and owing to the disadvantages of this method we develop a new factorization technique based on weighted ALS. Within the ALS algorithm we have two types of initialization methods, hence leading to ALS-I and ALS-II. The table below shows the various models built by combining the different biasing and factorizing techniques mentioned.

SVD

While using this method, which is [U,S,V]=svds(R,n), where R is the Rating matrix and n is the rank of the approximatin matrix. We test the models for different values of n, the maximum being n=200. The predictions are made in the following way,

	SVD	ALS-I	ALS-II
None	Model 1	Model 6	Model 7
$NonTemp_{mu}$	Model 3	-	Model 8
$Temp_{mu}$	Model 2	-	Model 9
$NonTemp_b$	-	Model 4	-
$Temp_b$	-	Model 5	-

Table 5.1. Different Model

$$[U, S, V] = svds(R, 200)$$
$$P_u = U(:, 1 : 200);$$
$$Q_i = S(1 : l, 1 : l) * Vt(1 : 200, :);$$
$$Predicted_rating(i, j) = P_u(i, :) * Q_i(:, j);$$

SVD technique

ALS-I

This is the type-I ALS method, where the M matrix is initialized by replacing the assigning the first row with the mean values of the movies, and filling in random values at the remaining places.

Step1. *Initialize the M matrix, by assigning the means of movies as the first row entry, and small random numbers in the remaining entries.*

Step2. *With this M, solve for U in order to minimize the loss function.*

Step3. *Now Keep the U matrix from step 2 fixed, and update the M matrix.*

Step4. *Repeat these iterations until required convergence criteria is met.*

ALS-II

This is the type-II ALS method, where the M matrix is initialized by doing an SVD on the initial rating matrix R, as shown below:

$$[U1, S1, V1] = svds(R_Hat_bin, 100)$$
$$M = S * V'$$

SVD Initialization of M

Step1. $Initialize the M matrix as shown above$

$row entry, and small random numbers in the remaining entries.$

Step2. $With this M, solve for U in order to minimize the loss function.$

Step3. $Now Keep the U matrix from step2 fixed, and update the M matrix.$

Step4. $Repeat these iterations until required convergence criteria is met.$

$NonTemp_{mu}$

This is the non-temporal model with the mean of all movies included in it, it is given by

$$b_{ui} = \mu + b_u + b_i \tag{5.1}$$

where b_{ui} is the baseline predictor, which encapsulates the non-interacting parts of user-item relationship, μ is the mean of all movies which is 3.6043, b_u and b_i are the observed deviations from the mean.

$Temp_{mu}$

This is the baseline predictor which involves the tie-changing movie biases, $b_{i,Bin(t)}$.

$$b_{ui} = \mu + b_u + b_i + b_{i,Bin(t)} \tag{5.2}$$

$NonTemp_b$

This is again non-temporal model, but without the all movie mean μ. This model is only used in conjuction with the ALS-I method, as in this method the means is already included into the model during the initialization of M.

$$b_{ui} = b_u + b_i \tag{5.3}$$

$Temp_b$

This is the temporal model without the all movie mean μ, again used for ALS-I but with time-changing movie biases,

$$b_{ui} = b_u + b_i + b_{i,Bin(t)} \tag{5.4}$$

5.2   Results

In the following figure 5.1, we can see that beyond certain ranks overfitting occurs. Also we cab observe the improvement in the results by involving the biases.

Next we show the results for the ALS based models. Figure 5.2 shows the RMSE of ALS based models which use the type I initialization technique.
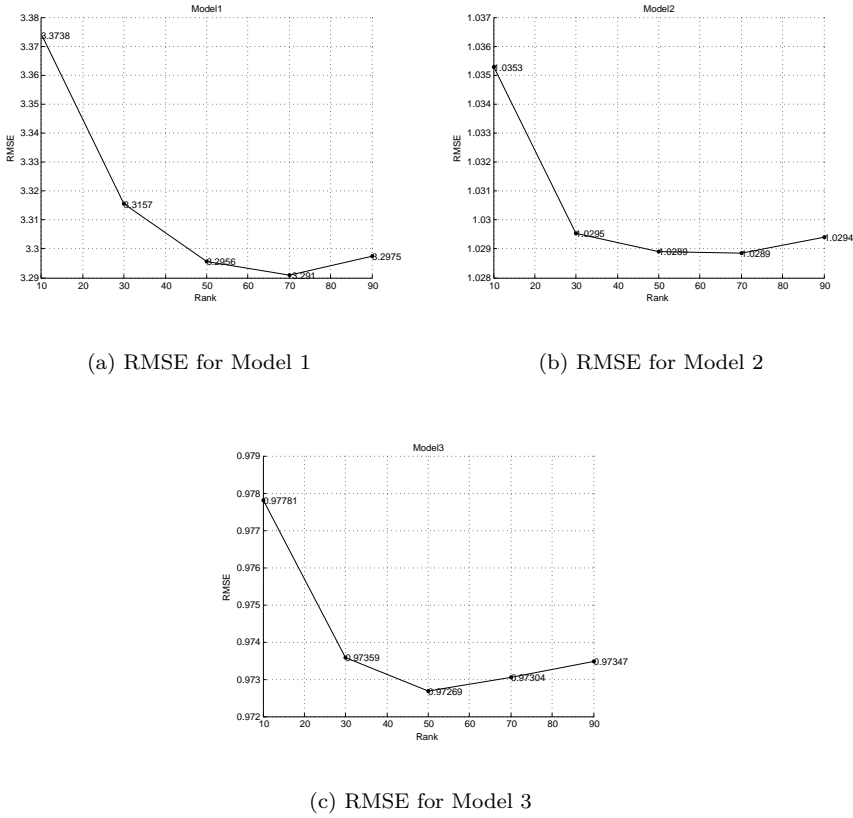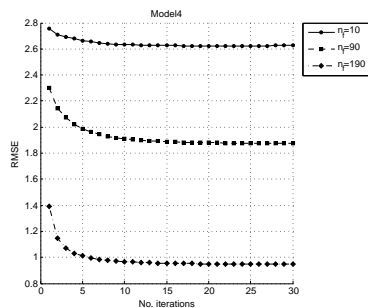
(a) RMSE for Model 1

(b) RMSE for Model 2

(c) RMSE for Model 3

Figure 5.1. RMSE: SVD based models

5.3   Future Work

In this work we have only considered only movie related temporal effects. We could for future work also include the more complex user realted temporal effects. The user related temporal dynamics proves to be more comples and more challenging to model. In a single household different people could be involved in rating movies, and the mood and temperment of the user is susceptible to more frequent changes. The user rates on forty different days on average. Apart from the mood swings, the user could also alter the rating scale over time. We could for further work consider the various user related effects and incude them into the model in order to get more accurate predictions.

Owing to the disadvantages of the sparse SVD of MATLAB, we have used an alternative method. Instead it is possible to develop a factorization technique based on SVD, but which is regularised. This approach was suggested initially by Simon Funk [17].

We could also aim to improve the prediction accuracy by considering inplicit feedback. By doing so we can take advantage of additional information about user preferences. In the current system, where we use only explicit ratings, where there are users for whom we have very little information, implicit data could prove to be useful [14].
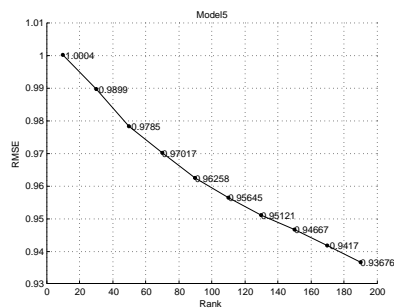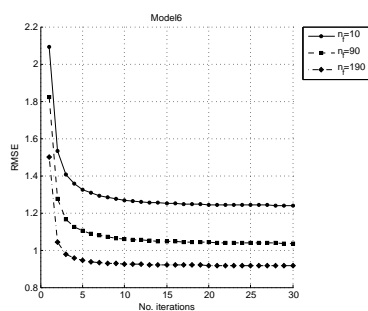
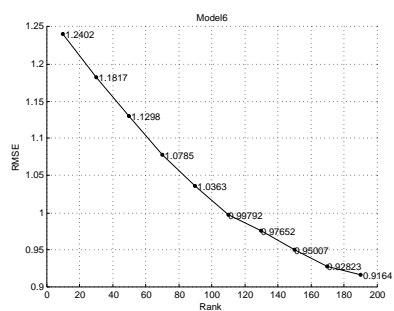(a) RMSE for Model 4 for 30 iterations $N_f = 200$
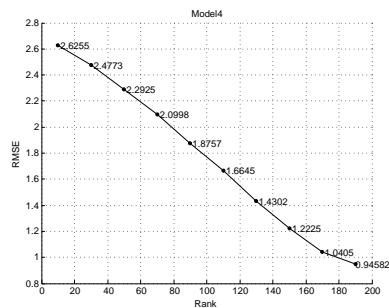
(b) RMSE for Model 4
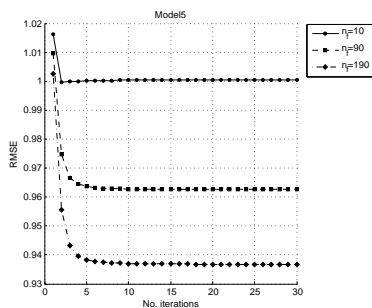
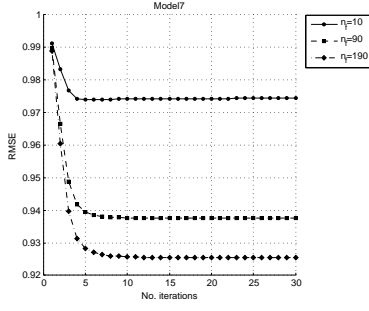(c) RMSE for Model 5 for 30 iterations $N_f = 200$

(d) RMSE for Model 5
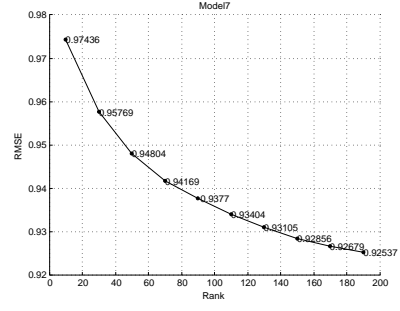
(e) RMSE for Model 6 for 30 iterations $N_f = 200$
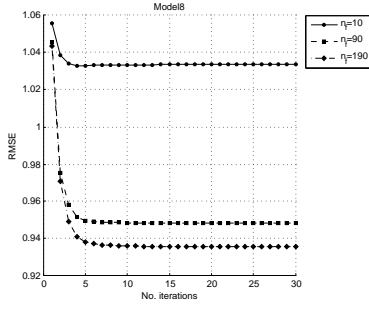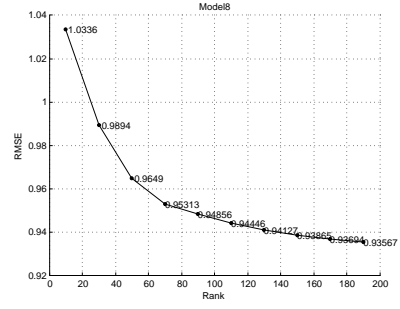
(f) RMSE for Model 6

Figure 5.2. RMSE: ALS-I based models

(a) RMSE for Model 7 for 30 iterations
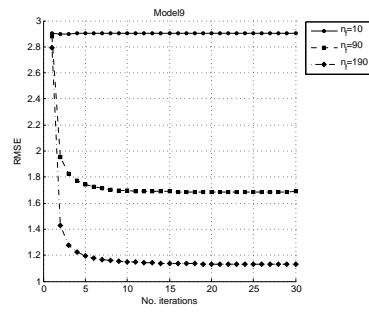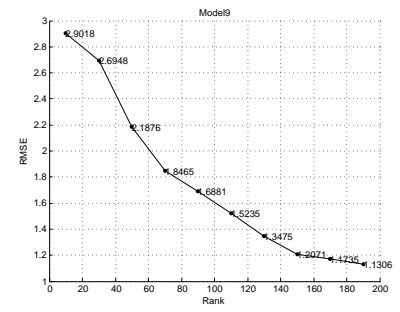$N_f = 200$

(b) RMSE for Model 7

(c) RMSE for Model 8 for 30 iterations
$N_f = 200$

(d) RMSE for Model 8

(e) RMSE for Model 9 for 30 iterations
$N_f = 200$

(f) RMSE for Model 9

Figure 5.3. RMSE: ALS-II based models

Bibliography

[1] How obama's data crunchers helped him win, http://edition.cnn.com/2012/11/07/tech/web/obama-campaign-tech-team/index.html.

[2] Netflix prize leaderboard, http://www.netflixprize.com//leaderboard.

[3] What is big data, ibm, http://www.ibm.com/big-data/us/en/.

[4] I. R. Edwards S. Olsson R. Orre A. Lansner R. M. De Freitas A. Bate, M. Lindquist. A bayesian neural network method for adverse drug reaction signal generation. *European Journal of Clinical Pharmacology*, 54(4).

[5] James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[6] Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 485–492, New York, NY, USA, 2005. ACM.

[7] Lars Eldén. *Matrix Methods in Data Mining and Pattern Recognition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007.

[8] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[9] Simon Funk. Netflix Update: Try this at Home. 2006.

[10] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.

[11] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[12] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[13] Yehuda Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, April 2010.

[14] Yehuda Koren and Robert Bell. Advances in Collaborative Filtering. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, chapter 5, pages 145–186. Springer US, Boston, MA, 2011.

[15] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[16] Jun Liu, Songcan Chen, Zhi-Hua Zhou, and Xiaoyang Tan. Generalized low-rank approximations of matrices revisited. *Trans. Neur. Netw.*, 21(4):621–632, April 2010.

[17] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering.

[18] Dirk deRoos Thomas Deutsch George Lapis Paul C.Zikopoulos, Chris Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data.* McGraw Hill.

[19] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, mar 1997.

[20] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th international conference on Algorithmic Aspects in Information and Management*, AAIM '08, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.

Appendix A

Sparse SVD Computation

SVDS

This function in MATLAB computes the singular vectors and singular values of a given sparse matrix.

Syntax

$[U, S, V] = svds(A, k)$,
where U is the *right singular vectors* $m \times k$ matrix having orthnormal columns,
V is the *left singular vectors* $n \times k$ matrix having orthnormal columns,
S is the $k \times k$ diagonal matrix, where the diagonal elements are the singular values.

Description

The MATLAB function svds uses Lanczos Tridiagonalization on the matrix,

$$B = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

Algorithm 1   Lanczos tridiagonalization
      Initialization: Starting vector v_1, such that $||v_1||_2 = 1$, $\beta_0 = 0$ and $v_0 = 0$.
  for $j = 1, 2, ...$ **do**
    $\alpha_j = v_j^T A v_j$.
    $v = A v_j - \alpha_j v_j - \beta_{j-1} v_{j-1}$
    $\beta_j = ||v||_2$
    $v_{j+1} = (1/\beta_j)v$
  end for
  end