A SINGULAR VALUE DECOMPOSITION APPROACH FOR RECOMMENDATION SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

OSMAN NURİ OSMANLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY 2010

Approval of the thesis:

A SINGULAR VALUE DECOMPOSITION APPROACH FOR RECOMMENDATION SYSTEMS

submitted by **OSMAN NURİ OSMANLI** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen ________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı ________________
Head of Department, **Computer Engineering**

Prof. Dr. İsmail Hakkı Toroslu

Supervisor, **Computer Engineering Dept., METU** ________________

Examining Committee Members:

Prof. Dr. Göktürk Üçoluk
Computer Engineering Dept., METU ______________________

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Dept., METU ______________________

Assist. Prof. Dr. Tolga Can
Computer Engineering Dept., METU _____________________

Assist. Prof. Dr. Pınar Şenkul
Computer Engineering Dept., METU ______________________

Güven Fidan
AGMLab ______________________

Date: 02.07.2010

ABSTRACT

A SINGULAR VALUE DECOMPOSITION APPROACH FOR RECOMMENDATION SYSTEMS

Osmanlı, Osman Nuri
M.Sc., Department of Computer Engineering
Supervisor: Prof. Dr. İsmail Hakkı TOROSLU

July 2010, 67 pages

Data analysis has become a very important area for both companies and researchers as a consequence of the technological developments in recent years. Companies are trying to increase their profit by analyzing the existing data about their customers and making decisions for the future according to the results of these analyses. Parallel to the need of companies, researchers are investigating different methodologies to analyze data more accurately with high performance.

Recommender systems are one of the most popular and widespread data analysis tools. A recommender system applies knowledge discovery techniques to the existing data and makes personalized product recommendations during live customer interaction. However, the huge growth of customers and products especially on the internet, poses some challenges for recommender systems, producing high quality recommendations and performing millions of recommendations per second.

In order to improve the performance of recommender systems, researchers have proposed many different methods. Singular Value Decomposition (SVD) technique based on dimension reduction is one of these methods which produces high quality recommendations, but has to undergo very expensive matrix calculations. In this thesis, we propose and experimentally validate some contributions to SVD technique which are based on the user and the item categorization. Besides, we adopt tags to classical 2D (User-Item) SVD technique and report the results of experiments. Results are promising to make more accurate and scalable recommender systems.

Keywords: Recommender Systems, Collaborative Filtering, Singular Value Decomposition, Content Based Filtering, Personalization, User Modeling

ÖZ

ÖNERİ SİSTEMLERİ İÇİN TEKİL DEĞER AYRIŞIMI YAKLAŞIMI

Osmanlı, Osman Nuri
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. İsmail Hakkı TOROSLU

Temmuz 2010, 67 sayfa

Son yıllarda meydana gelen teknolojik gelişmeler sonucu, veri analizi şirketler ve araştırmacılar açısından son derece önemli bir alan haline gelmektedir. Şirketler, müşterileri ile ilgili ellerinde varolan bilgileri analiz ederek ve ileriki kararlarını alırken bu analizlerine göre hareket ederek karlarını artırmaya çalışmaktadır. Şirketlerin bu ihtiyaçlarına paralel olarak, araştırmacılar veriyi daha doğru ve hızlı işleyebilmek için farklı metodolojiler geliştirmektedir.

Öneri sistemleri bu açılardan en popüler ve en yaygın veri analiz yazılımlarıdır. Bir öneri sistemi mevcut veriye bilgi işleme tekniklerini uygulayıp analiz ederek, kullanıcılarına kisiselleştirilmiş ürün önerileri sunar. Ancak özellikle internetin yaygınlaşması ile müşteri ve ürün sayısındaki büyük artışlar öneri sistemleri için bazı problemleri beraberinde getirmektedir. Bu problemler yüksek kalitede önerilerin yapılması ve saniyede milyonlarca öneri isteğine cevap verebilmektir.

Öneri sistemlerinin performansını artırmaya yönelik, araştırmacılar tarafından birçok metod önerilmektedir. Boyut indirgemeye dayalı Tekil Değer Ayrışımı (TDA), son derece yüksek kalitede öneriler üreten fakat hesaplama açısından pahalı matris işlemleri gerektiren bir yöntemdir. Bu tez kapsamında TDA'ya dayalı öneri tekneğine, kullanıcıya ve ürünlere göre kategoriler oluşturma, ve bu kategorileri öneri sürecine dahil etme işlemi önerilmekte ve bu eklemenin olumlu sonuçları deneyler ile doğrulanmaktadır. Bunun yanında, klasik iki boyutlu TDA tekniğine, üçüncü boyut olarak etiketler adapte edilmiş ve deneysel sonuçlar raporlanmıştır. Bu iyileştirmeler daha doğru ve genişletilebilir öneri sistemleri oluşturmayı sağlamıştır.

Anahtar Kelimeler: Öneri Sistemleri, Tekil Değer Ayrışımı, Kolaboratif Filtreleme, İçerik Bazlı Filtreleme, Kişiselleştirme, Kullanıcı Modelleme

To my family

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to Prof. İsmail Hakkı Toroslu for his encouragement and support throughout this study.

I am deeply grateful to my wife and my family for their love and support. Without them, this work could never have been completed.

I would thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing the financial means throughout this study.

Finally, my special thanks go to my friends, Fatih Deniz, Erkay Uzun, Fatih Gökbayrak, Metehan Aydın, Derya Susman, and Sermet Reha Yavuz for their help, support and cheerful presence through not only the course of this study but also any course of my life. Thanks for giving me a shoulder to lean on, whenever I need.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RS	Recommender Systems
IR	Information Retrieval
IF	Information Filtering
CB	Content Based
CF	Collaborative Filtering
SVD	Singular Value Decomposition
UU	User to User Similarity
II	Item to Item Similarity

CHAPTER 1

INTRODUCTION

Recommender systems apply data analysis techniques to the problem of helping customers to find which products they would like to purchase especially on the internet. These systems are rapidly becoming a crucial tool in E-commerce on the Web. The tremendous growth of customers and products poses two main challenges for recommender systems. The first challenge is to improve the quality of the recommendations for the customers. Making good recommendations increases the customers desire to purchase products, whereas making bad recommendations may result losing customers. Another challenge is to improve the scalability of the recommendation algorithms. These algorithms are able to respond tens of millions of recommendation requests in real-time. In order to make a system scalable, the response time for the requests should be reduced. However, if the algorithm spends less time for recommendation, the quality of the recommendation decreases. Actually, from this perspective these two challenges are in conflict. For this reason, it is important to consider both of them simultaneously for the proposed solutions.

Every recommendation system follows a specific process while making recommendations. Systems use the users' profiles and the information about items or products as the inputs and produce recommendations. In other words, a recommendation system consists of background data, the information that the system has before the recommendation process begins, input data, the information that user must communicate to the system in order to generate a recommendation, and an algorithm that combines background and input data to arrive at its recommendations.

Recommendation techniques can be grouped into five as collaborative, content-based, demographic, utility-based, and knowledge-based [11]. Moreover, some hybrid solutions could be generated with different methods as a combination of some of these five techniques. In Chapter 2, all of these techniques are explained briefly. Among several different techniques, collaborative recommendation is probably the most familiar, most widely implemented one. Collaborative recommender systems recognize commonalities between users on the basis of their ratings, and generate new recommendations based on inter-user comparisons.

At this point, SVD has an important property that makes it interesting for recommender systems. SVD provides the best low-rank linear approximation of the original matrix and the low-rank approximation of the original matrix is better than the original matrix itself [1, 2]. Filtering out of the small singular values can be introduced as removing "noise" data in the matrix. Researchers [2, 3, 4, 5] suggest that SVD-based approaches produce results better than traditional collaborative filtering algorithms most of the time. However, SVD requires computationally very expensive matrix calculations and this makes SVD-based recommender systems less suitable for large-scale systems. For this reason, most of the researches on SVD-based recommendation focus on scalability problem while protecting the high quality recommendations of the method.

In this thesis, SVD-based recommendation techniques are compared with experiments and some new approaches are introduced to this technique. The first contribution we have proposed is the categorization of items and users. Our experiments showed that, item and user categorization increases both the recommendation quality and speed performance of the SVD technique. Moreover, we adopted the tags to the traditional 2-Dimensional SVD approach. By this way, we have the chance to analyze the effect of third dimension (tags) to the SVD recommendation performance. Our experiments illustrated that, tags also increase the performance to some extent.

This thesis is structured in the following way:

In chapter 2, a detailed explanation about recommender systems and a more formal description of the recommendation process is presented.

Chapter 3 covers the SVD method with examples. In addition to this chapter, Appendix A provides background information for SVD including basic matrix terminology.

In Chapter 4, the existing SVD recommendation approaches are presented. The algorithms are explained in details. In addition to the existing methods we report our proposals, the categorization of users and items in SVD recommendation and adopting tags to classical SVD approach.

In Chapter 5, the experimental results for both existing algorithms and our proposals are demonstrated together with the comparisons and evaluations about the results.

Chapter 6 draws the conclusions of this thesis work. Other than that, some possible future work direction in terms of both our approach and the related area are stated.

CHAPTER 2

RECOMMENDER SYSTEMS

2.1 Definition of a Recommender System (RS)

In order to increase the users' satisfaction towards online information search results, search engine developers and vendors try to predict user preference based on the user behavior. Recommendations are provided by the search engines or online vendors to the users. Recommendation systems are implemented in both commercial and non-profit web sites to predict the user preferences. Accurate predictions may result in higher selling rates and increase the customer satisfaction. The main functions of recommendation systems are analyzing user data and extracting useful information for further predictions [6]. Variety of techniques has been proposed for performing recommendation, including content-based, collaborative, knowledge-based and other techniques. To improve performance, these methods have sometimes been combined in hybrid recommenders.

There are a lot of recommendation systems, accessible via internet, which attempt to recommend to users several products such as music, movies, books, etc. For instance, recommender systems are now an integral part of some e-commerce sites such as Amazon.com and CDNow [7]. In a general way, recommendation systems are systems which intend to acquire opinions or preferences about items from a community of users, and use those opinions to present other users with items that are interesting to them. From this general description we can see that recommendation systems need two basic things to work properly:

1. Information about the preferences of the users
2. A method to determine if an item is interesting for a user

Normally, the users' information includes external information, such as user profiles, purchasing histories, and product ratings [8]. The way to determine whether an item is interesting to a user or not, depends on the kind of recommendation system. In this chapter, we will discuss the recommendation techniques which are commonly used.

2.1.1 Recommendation Process

In general, every recommendation system follows a specific process in order to create recommendations. If we see the process of recommendation as a black box, as shown in Figure 2.1, we can identify two sources of information needed as input for the process. These sources of information are the users' profiles and the information about items or products. Ideally the information stored in the profiles is related with the preferences of the users and should be given explicitly by the user itself. However, this information can also be extracted from other external sources such as web pages, buying behavior, etc.

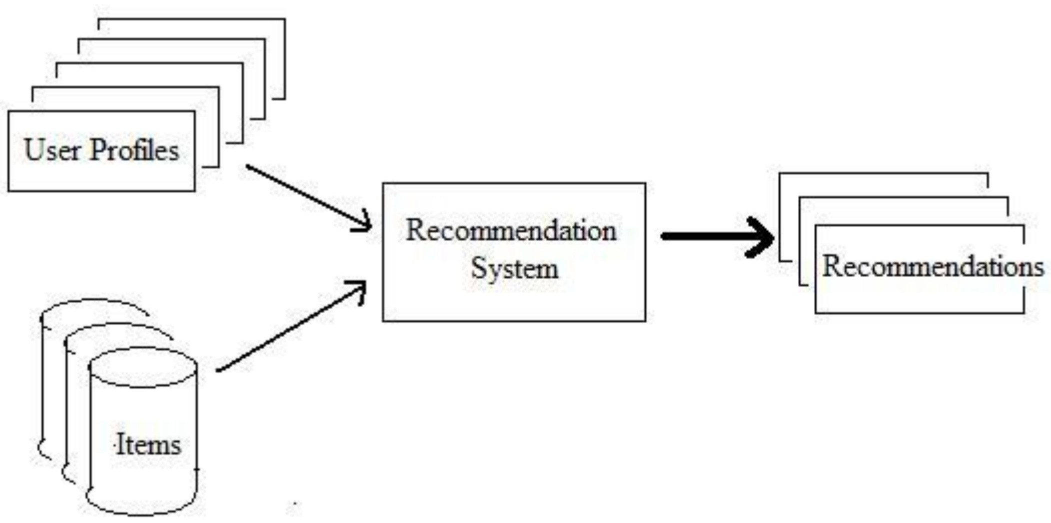


Figure 2.1 - Recommendation process as a black box

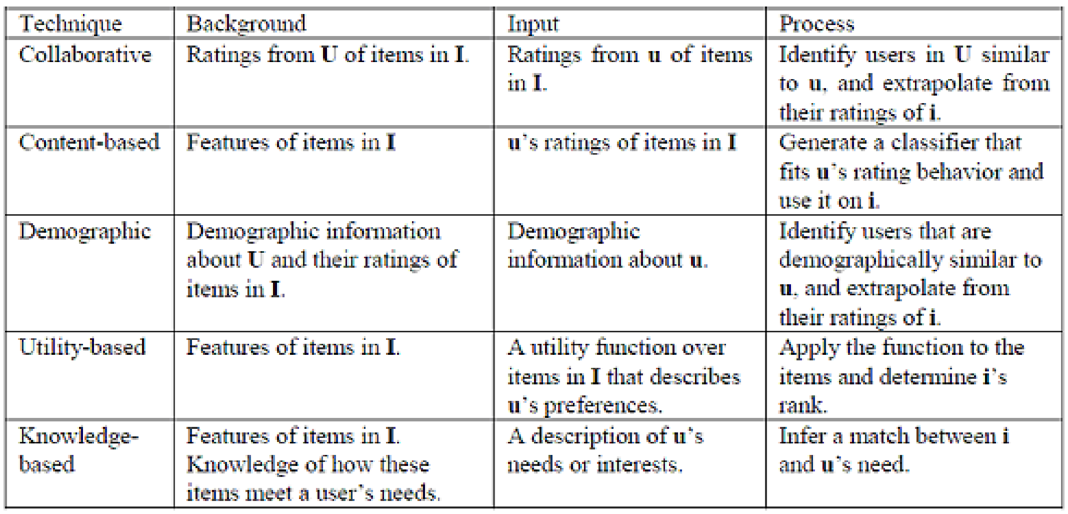
2.2 Recommendation Techniques

Recommendation techniques have a number of possible classifications [7, 9, 10]. Specifically, recommender systems have

- background data, the information that the system has before the recommendation process begins
- input data, the information that user must communicate to the system in order to generate a recommendation
- algorithm that combines background and input data to arrive at its suggestions.

On this basis, recommendation techniques can be grouped into five as shown in Table 2.1 [11]. In the table **I** is the set of items over which recommendations might be made, **U** is the set of users whose preferences are known, **u** is the user for whom recommendations need to be generated, and **i** is the item for which we would like to predict **u**'s preference.

Table 2.1 - Recommendation Techniques



Technique	Background	Input	Process
Collaborative	Ratings from **U** of items in **I**.	Ratings from **u** of items in **I**.	Identify users in **U** similar to **u**, and extrapolate from their ratings of **i**.
Content-based	Features of items in **I**	**u**'s ratings of items in **I**	Generate a classifier that fits **u**'s rating behavior and use it on **i**.
Demographic	Demographic information about **U** and their ratings of items in **I**.	Demographic information about **u**.	Identify users that are demographically similar to **u**, and extrapolate from their ratings of **i**.
Utility-based	Features of items in **I**.	A utility function over items in **I** that describes **u**'s preferences.	Apply the function to the items and determine **i**'s rank.
Knowledge-based	Features of items in **I**. Knowledge of how these items meet a user's needs.	A description of **u**'s needs or interests.	Infer a match between **i** and **u**'s need.

2.2.1 Collaborative Recommendation

Collaborative recommendation is probably the most well known, most widely implemented and most mature of the technologies. Collaborative recommender systems aggregate ratings or recommendations of objects, recognize commonalities between users on the basis of their ratings, and generate new recommendations based on inter-user comparisons. A typical user profile in a collaborative system consists of a vector of items and their ratings. In some cases, ratings may be binary (like/dislike) or real-valued indicating degree of preference. Some of the most important systems using this technique are GroupLens/NetPerceptions [12], Ringo/Firefly [13], and Recommender [14]. According to [15], these systems can be:

- memory-based, comparing users against each other directly using correlation or other measures

- model-based, in which a model is derived from the historical rating data and used to make predictions

2.2.1.1 Memory-based Algorithms

These are algorithms which make predictions based on the entire collection of previously rated items by the users. That is, the value of the unknown rating for a user and an item is usually computed as an aggregate of the ratings of some other (usually the N most similar) users for the same item [16].

Memory-based approaches are the most popular prediction methods and are widely adopted in commercial collaborative filtering systems. The major types of memory-based approaches are:

- **User-based Approaches**

  User-based Collaborative Filtering predicts an active user’s interest in a particular item based on rating information from similar user profiles, where each user profile corresponds to a row vector sorted in the user-item matrix. First, all similarities of any two row vectors are calculated. Then, for predicting the rating of a user for a particular item, a set of top-N similar users are identified. The ratings of those top-N users are averaged as the prediction by weighted [17].

These systems are extremely data-intensive, typically requiring a large number of user ratings before they can make reasonable recommendations. Moreover, depending on how actively a user rates content, the system may be slow to accumulate enough information about a user's preferences to make accurate recommendations, resulting in poor recommendations for a prolonged period [18].

- **Item-based Approaches**

These approaches use the similarity between items instead of users. After, the similarity of items (column vectors in the user-item matrix) are calculated, unknown ratings can be predicted by averaging the ratings of other similar items rated by the active user [17].

The main advantage of item-based collaborative filtering over user-based collaborative filtering is its scalability. Item-based collaborative filtering does not have to scour databases containing potentially millions of users in order to find users with similar tastes. Instead, it can pre-score content based on user ratings and/or their attributes, and then make recommendations without incurring high computation costs [18].

2.2.1.2 Model-based Algorithms

In contrast to memory-based algorithms, model-based algorithms use the collection of ratings as a training dataset to learn a model, which is then used to make rating predictions [16]. The model-based approaches are often time-consuming to build and update, and cannot cover a user range as diverse as the memory approaches [19].

Model-based recommenders have used a variety of learning techniques including neural networks [20], latent semantic indexing [21], and Bayesian networks [22].

2.2.2 Content-based Recommendation

Content-based recommendation is an outgrowth and continuation of information filtering research [23]. In a content-based system, the objects of interest are defined by their associated features. For example, text recommendation systems like the newsgroup filtering system NewsWeeder [24] uses the words of their texts as features. A content-based recommender learns a profile of the user's interests based on the features present in objects the user has rated. The type of user profile derived by a content-based recommender depends on the learning method employed. Decision trees, neural nets, and vector-based representations have all been used. As in the collaborative case, content-based user profiles are long-term models and updated as more evidence about user preferences is observed.

2.2.3 Demographic Recommendation

Demographic recommender systems aim to categorize the user based on personal attributes and make recommendations based on demographic classes. An early example of this kind of system was Grundy [25] that recommended books based on personal information gathered through an interactive dialogue. Some more recent recommender systems have also taken this approach. For example, [26] uses demographic groups from marketing research to suggest a range of products and services. A short survey is used to gather the data for user categorization. The representation of demographic information in a user model can vary greatly. Demographic techniques form "people-to-people" correlations like collaborative ones, but use different data. The benefit of a demographic approach is that it may not require a history of user ratings of the type needed by collaborative and content-based techniques.

2.2.4 Utility-based Recommendation

Utility-based recommenders make suggestions based on a computation of the utility of each object for the user. Of course, the central problem is how to create a utility function for each user. Tête-à-Tête and the e-commerce site PersonaLogic2 each have different techniques for arriving at a user-specific utility function and applying it to the objects under consideration [27]. The user profile therefore is the utility function that the system has derived for the user, and the system employs constraint satisfaction techniques to locate the best match. The benefit of utility-based recommendation is that it can factor non-product attributes, such as vendor reliability and product availability, into the utility computation, making it possible for example to trade off price against delivery schedule for a user who has an immediate need.

2.2.5 Knowledge-based Recommendation

Knowledge-based recommendation attempts to suggest objects based on inferences about a user's needs and preferences. In some sense, all recommendation techniques could be described as doing some kind of inference. Knowledge-based approaches are distinguished in that they have functional knowledge:

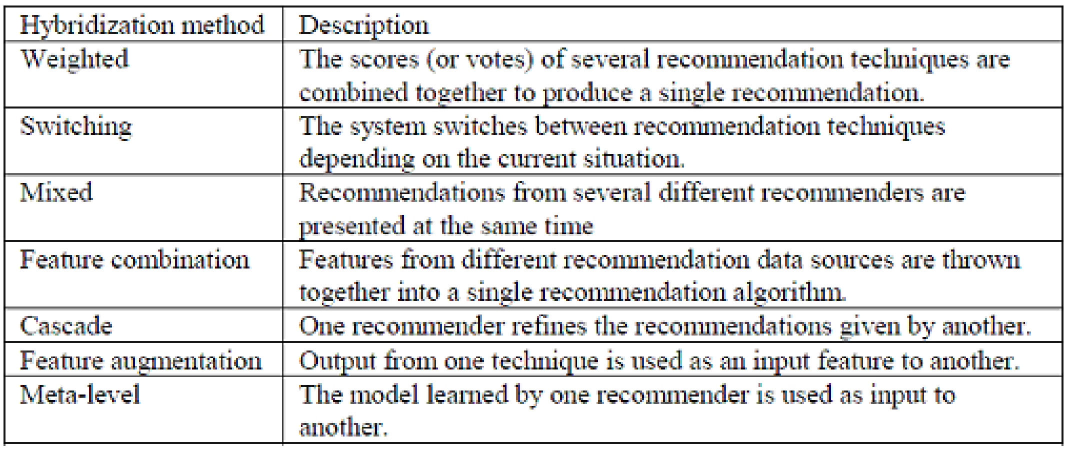
- how a particular item meets a particular user need
- reason about the relationship between a need and a possible recommendation

The user profile can be any knowledge structure that supports this inference. In the simplest case, as in Google, it may simply be the query that the user has formulated. In others, it may be a more detailed representation of the user's needs [28].

2.3 Hybrid Recommendation Systems

Hybrid recommender systems combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one. Most commonly, collaborative filtering is combined with some other technique in an attempt to avoid the ramp-up problem. Table 2.3 shows some of the combination methods that have been employed [11].

Table 2.2 - Hybridization Methods



Hybridization method	Description
Weighted	The scores (or votes) of several recommendation techniques are combined together to produce a single recommendation.
Switching	The system switches between recommendation techniques depending on the current situation.
Mixed	Recommendations from several different recommenders are presented at the same time
Feature combination	Features from different recommendation data sources are thrown together into a single recommendation algorithm.
Cascade	One recommender refines the recommendations given by another.
Feature augmentation	Output from one technique is used as an input feature to another.
Meta-level	The model learned by one recommender is used as input to another.

- **Weighted:** Each of the recommendation approaches that makes predictions are combined into a single prediction. For example, the simplest combined hybrid would be a linear combination of recommendation scores. The P-Tango system uses such a hybrid [32].

- **Switching:** One of the recommendation techniques is selected to make the prediction when certain criteria are met. The DailyLearner system uses a content/collaborative hybrid in which a content-based recommendation method is employed first. If the content-based system cannot make a recommendation with sufficient confidence, then a collaborative recommendation is attempted [11].
- **Mixed:** Predictions from each of the recommendation techniques are presented to the user. The PTV system uses this approach to assemble a recommended program of television viewing [33]. It uses content-based techniques based on textual descriptions of TV shows and collaborative information about the preferences of other users. Recommendations from the two techniques are combined together in the final suggested program.
- **Feature Combination:** A single prediction algorithm is provided with features from different recommendation techniques. For example, [34] report on experiments in which the inductive rule learner Ripper was applied to the task

of recommending movies using both user ratings and content features, and achieved significant improvements in precision over a purely collaborative approach.

- **Cascade:** Output from one recommendation technique is refined by another. Unlike the previous hybridization methods, the cascade hybrid involves a staged process. In this technique, on recommendation technique is employed first to produce a coarse ranking of candidates and a second technique refines the recommendation from among the candidate set [11].

- **Feature Augmentation:** Output from one recommendation technique is fed to another. One technique is employed to produce a rating or classification of an item and that information is then incorporated into the processing of the next recommendation technique. For example, the Libra system makes content-based recommendations of books based on data found in Amazon.com, using a naive Bayes text classifier. In the text data used by the system is included “related authors” and “related titles” information that Amazon generates using its internal collaborative systems [35].

- **Meta-level:** Entire model produced by one recommendation technique is utilized by another. This differs from feature augmentation: in an augmentation hybrid, we use a learned model to generate features for input to a second algorithm; in a meta-level hybrid, the entire model becomes the input. The first meta-level hybrid was the web filtering system Fab [36].

CHAPTER 3

SINGULAR VALUE DECOMPOSITION

In this section, SVD is explained with examples. Examples are taken from the Singular Value Decomposition tutorial [37]. The background information about matrix basics, Eigenvalues and Eigenvectors is available in Appendix A.

3.1 Singular Value Decomposition (SVD)

3.1.1 Definition of SVD

Singular value decomposition (SVD) can be seen as a method for data reduction. As an illustration of this idea, consider the 2-dimensional data points in Figure 3.1 [37].

The regression line running through them shows the best approximation of the original data with a 1-dimensional object (a line). It is the best approximation in the sense that it is the line that minimizes the distance between each original point and the line.

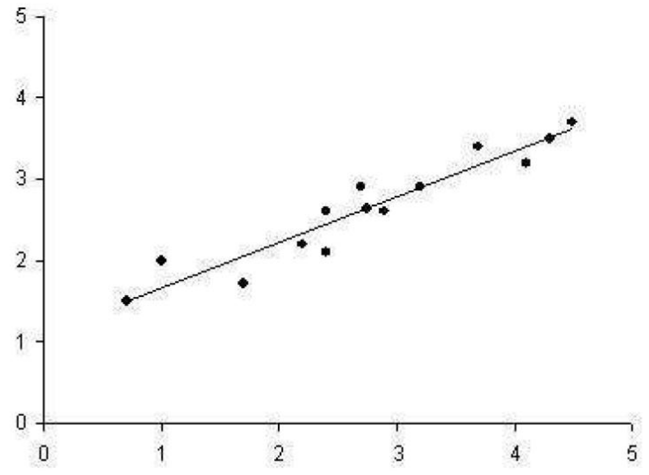


Figure 3.1 - Best-fit regression line

If we drew a perpendicular line from each point to the regression line, and took the intersection of those lines as the approximation of the original data point, we would have a reduced representation of the original data that captures as much of the original variation as possible. Notice that there is a second regression line, perpendicular to the first, shown in Figure 3.2 [37].

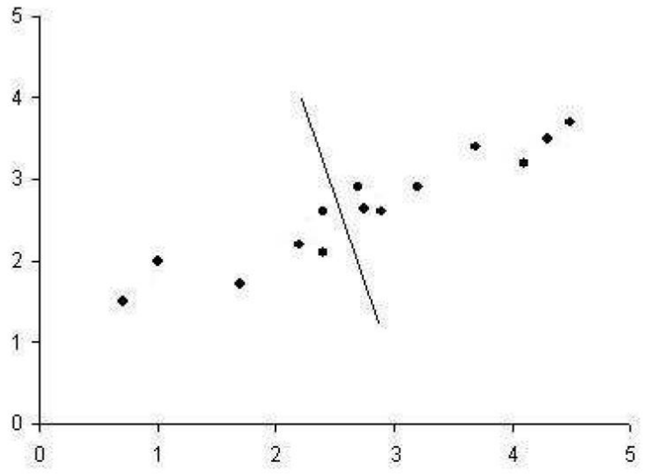


Figure 3.2 - Regression line along second dimension

This line captures as much of the variation as possible along the second dimension of the original data set. It does poorer job of approximating the original data because it corresponds to a dimension exhibiting less variation to begin with. It is possible to use these regression lines to generate a set of uncorrelated data points that will show sub groupings in the original data not necessarily visible at first glance.

These are the basic ideas behind SVD: taking a high dimensional, highly variable set of data points and reducing it to a lower dimensional space that exposes the substructure of the original data more clearly and orders it from most variation to the least.

Singular Value Decomposition is a matrix factorization technique which takes a rectangular matrix defined as A where A is an *m x n* matrix in which the *m* rows represents the users, and the *n* columns represents the items. The SVD theorem (1) states:

$$\mathbf{A}_{mxn} = \mathbf{U}_{mxm}\,\mathbf{S}_{mxn}\,\mathbf{V}^{\mathbf{T}}_{nxn} \tag{1}$$

Where $\mathbf{U}^T\mathbf{U} = \mathbf{I}_{mxm}$

$$\mathbf{V}^T\mathbf{V} = \mathbf{I}_{nxn}$$

Where the columns of U are the left singular vectors; S (the same dimensions as *A*) has singular values and is diagonal; and V^T has rows that are the right singular vectors. Calculating the SVD consists of finding the Eigenvalues and Eigenvectors of $\mathbf{AA^T}$ and $\mathbf{A^TA}$. The Eigenvectors of $\mathbf{A^TA}$ make up the columns of **V** , the Eigenvectors of $\mathbf{AA^T}$ make up the columns of **U**. Also, the singular values in **S** are square roots of Eigenvalues from $\mathbf{AA^T}$ or $\mathbf{A^TA}$. The singular values are the diagonal entries of the **S** matrix and are arranged in descending order. The singular values are always real numbers. If the matrix **A** is a real matrix, then **U** and **V** are also real.

Matrix **S** is a diagonal matrix having only *r* nonzero entries, which makes the effective dimensions of **U**, **S** and **V** matrices $m \times r$, $r \times r$, and $r \times n$, respectively. The diagonal entries ($s_1, s_2, \ldots, s_r$) of *S* have the property that $s_i > 0$ and $s_1 \geq s_2 \geq \ldots \geq s_r$.

3.1.2 Example of SVD

Start with the matrix

$$A = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

In order to find U, we have to start with AA^T. The transpose of A is

$$A^T = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix}$$

So

$$AA^T = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$$

Next, we have to find the Eigenvalues and corresponding Eigenvectors of AA^T. We know that Eigenvectors are defined by the equation

$$A\vec{v} = \lambda\vec{v}.$$

And applying this to AA^T gives us

$$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We rewrite this as the set of equations

$$11x_1 + x_2 = \lambda x_1$$

$$x_1 + 11x_2 = \lambda x_2$$

And rearrange to get

$$(11-\lambda)x_1 + x_2 = 0$$
$$x_1 + (11-\lambda)x_2 = 0$$

Solve for λ by setting the determinant of the coefficient matrix to zero,

$$\begin{vmatrix} (11-\lambda) & 1 \\ 1 & (11-\lambda) \end{vmatrix} = 0$$

This works out as

$$(11-\lambda)(11-\lambda) - 1\cdot 1 = 0$$
$$(\lambda-10)(\lambda-12) = 0$$
$$\lambda = 10, \lambda = 12$$

To give us our two Eigenvalues λ = 10; λ = 12. Plugging λ back in to the original equations gives us our Eigenvectors.

For λ = 10, we get

$$(11-10)x_1 + x_2 = 0$$

$$x_1 = -x_2$$

Which is true for lots of values, so we'll pick x_1 = 1 and x_2 = -1 since those are small and easier to work with. Thus, we have the Eigenvector [1; -1] corresponding to the Eigenvalue λ = 10.

For λ = 12, we get

$$(11-12)x_1 + x_2 = 0$$

$$x_1 = x_2$$

And for the same reason as before we'll take $x_1 = 1$ and $x_2 = 1$. Now, for λ = 12 we have the Eigenvector [1; 1]. These Eigenvectors become column vectors in a matrix ordered by the size of the corresponding Eigenvalue. In other words, the Eigenvector of the largest Eigenvalue is column one, the Eigenvector of the next largest Eigenvalue is column two, and so forth and so on until we have the Eigenvector of the smallest Eigenvalue as the last column of our matrix. In the matrix below, the Eigenvector for λ = 12 is column one, and the Eigenvector for λ = 10 is column two.

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Finally, we have to convert this matrix into an orthogonal matrix which we do by applying the Gram-Schmidt orthonormalization process to the column vectors. Begin by normalizing v_1.

$$\vec{u_1} = \frac{\vec{v_1}}{|\vec{v_1}|} = \frac{[1,1]}{\sqrt{1^2+1^2}} = \frac{[1,1]}{\sqrt{2}} = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$$

Compute

$$\vec{w_2} = \vec{v_2} - \vec{u_1} \cdot \vec{v_2} * \vec{u_1} =$$

$$[1,-1] - [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}] \cdot [1,-1] * [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}] =$$

$$[1,-1] - 0 * [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}] = [1,-1] - [0,0] = [1,-1]$$

And normalize

$$\vec{u_2} = \frac{\vec{w_2}}{|\vec{w_2}|} = [\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]$$

To give

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$$

The calculation of V is similar. V is based on A^TA, so we have

$$A^TA = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

Find the Eigenvalues of A^TA by

$$\begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Represents the system of equations

$$10x_1 + 2x_3 = \lambda x_1$$

$$10x_2 + 4x_3 = \lambda x_2$$

$$2x_1 + 4x_2 + 2x_3 = \lambda x_2$$

Rewrite as

$$(10 - \lambda)x_1 + 2x_3 = 0$$

$$(10 - \lambda)x_2 + 4x_3 = 0$$

$$2x_1 + 4x_2 + (2 - \lambda)x_3 = 0$$

These are solved by setting

$$\begin{vmatrix} (10-\lambda) & 0 & 2 \\ 0 & (10-\lambda) & 4 \\ 2 & 4 & (2-\lambda) \end{vmatrix} = 0$$

This works out as

$$(10-\lambda)\begin{vmatrix} (10-\lambda) & 4 \\ 4 & (2-\lambda) \end{vmatrix} + 2\begin{vmatrix} 0 & (10-\lambda) \\ 2 & 4 \end{vmatrix} =$$

$$(10-\lambda)[(10-\lambda)(2-\lambda) - 16] + 2[0 - (20 - 2\lambda)] =$$
$$\lambda(\lambda - 10)(\lambda - 12) = 0,$$

so $\lambda = 0$, $\lambda = 10$, $\lambda = 12$ are the Eigenvalues for A^TA. Substituting λ back into the original equations to find corresponding Eigenvectors yields for $\lambda = 12$

$$(10-12)x_1 + 2x_3 = -2x_1 + 2x_3 = 0$$

$$x_1 = 1, x_3 = 1$$

$$(10-12)x_2 + 4x_3 = -2x_2 + 4x_3 = 0$$

$$x_2 = 2x_3$$

$$x_2 = 2$$

So for λ = 12, v_1 = [1; 2; 1].

For λ = 10, we have

$$(10-10)x_1 + 2x_3 = 2x_3 = 0$$

$$x_3 = 0$$

$$2x_1 + 4x_2 = 0$$

$$x_1 = -2x_2$$

$$x_1 = 2, x_2 = -1$$

Which means for λ = 10, v_2 = [2; -1; 0].

For λ = 0, we have

$$10x_1 + 2x_3 = 0$$

$$x_3 = -5$$

$$10x_1 - 20 = 0$$

$$x_2 = 2$$

$$2x_1 + 8 - 10 = 0$$

$$x_1 = 1$$

Which means for λ = 0, v_3 = [1; 2; -5]. Order v_1, v_2, and v_3 as column vectors in a matrix according to the size of the Eigenvalue to get and use the Gram-Schmidt orthonormalization process to convert that to an orthonormal matrix.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 2 \\ 1 & 0 & 5 \end{bmatrix}$$

$$\vec{u_1} = \frac{\vec{v_1}}{|\vec{v_1}|} = [\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}, \frac{1}{\sqrt{6}}]$$

$$\vec{w_2} = \vec{v_2} - \vec{u_1} \cdot \vec{v_2} * \vec{u_1} = [2, -1, 0]$$

$$\vec{u_2} = \frac{\vec{w_2}}{|\vec{w_2}|} = [\frac{2}{\sqrt{5}}, \frac{-1}{\sqrt{5}}, 0]$$

$$\vec{w_3} = \vec{v_3} - \vec{u_1} \cdot \vec{v_3} * \vec{u_1} - \vec{u_2} \cdot \vec{v_3} * \vec{u_2} = [\frac{-2}{3}, \frac{-4}{3}, \frac{10}{3}]$$

$$\vec{u_3} = \frac{\vec{w_3}}{|\vec{w_3}|} = [\frac{1}{\sqrt{30}}, \frac{2}{\sqrt{30}}, \frac{-5}{\sqrt{30}}]$$

All this to give us

$$V = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

When we really want its transpose

$$V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix}$$

For S we take the square roots of the non-zero Eigenvalues and populate the diagonal with them, putting the largest in s_{11}, the next largest in s_{22} and so on until the smallest value ends up in s_{mm}. The non-zero Eigenvalues of U and V are always the same, so that's why it doesn't matter which one we take them from. The diagonal entries in S are the singular values of A, the columns in U are called left singular vectors, and the columns in V are called right singular vectors.

$$S = \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix}$$

Now we have all the pieces of the puzzle

$$A_{mn} = U_{mm} S_{mn} V_{nn}^T =$$

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix} =$$

$$\begin{bmatrix} \frac{\sqrt{12}}{\sqrt{2}} & \frac{\sqrt{10}}{\sqrt{2}} & 0 \\ \frac{\sqrt{12}}{\sqrt{2}} & \frac{-\sqrt{10}}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

CHAPTER 4

RECOMMENDATION WITH SVD

4.1 Recommendation Using SVD

The goal of CF-based recommendation algorithms is to suggest new products or to predict the utility of a product for a customer, based on the customer's previous behaviour and other similar customers' opinions. However, these systems have some problems like sparsity, scalability, and synonymy. The weakness of CF algorithms for large, sparse databases led the researchers to alternative ways. In order to remove noise data from a large and sparse database, some dimensionality reduction techniques are proposed[1, 2, 3]. Latent Semantic Indexing (LSI), which is a dimensionality reduction technique that used in information retrieval (IR), is a widely used technique to reduce the dimensionality of user-item ratings matrix. LSI, which uses singular value decomposition (SVD) as its underlying dimension reduction algorithm, maps nicely into the collaborative filtering recommender algorithm challenge [1]. SVD-based recommendation algorithms produce high quality recommendations, but has to undergo computationally very expensive matrix factorization steps [1].

4.1.1 Dimensionality Reduction

SVD has an important property that makes it interesting for recommender systems. SVD provides the best low-rank linear approximation of the original matrix. It is possible to reduce dimensions by selecting greatest k singular values. The value of k may change according to the size and the structure of data.

The reduced matrix $\mathbf{S_k}$ is constructed by retaining the first $\boldsymbol{k}$ singular values. The matrices **U** and **V** are also reduced to produce matrices $\mathbf{U_k}$ and $\mathbf{V_k}$, respectively. The matrix $\mathbf{U_k}$ is produced by removing $(\boldsymbol{r-k})$ columns from the matrix **U** and matrix $\mathbf{V_k}$ is produced by removing $(\boldsymbol{r-k})$ rows from the matrix **V**. Multiplying these three reduced matrices, the matrix $\mathbf{A_k}$ is obtained. The reconstructed matrix $\mathbf{A_k}$ is a matrix that is the closest approximation to the original matrix *A*. Figure 3.3 demonstrates this process.

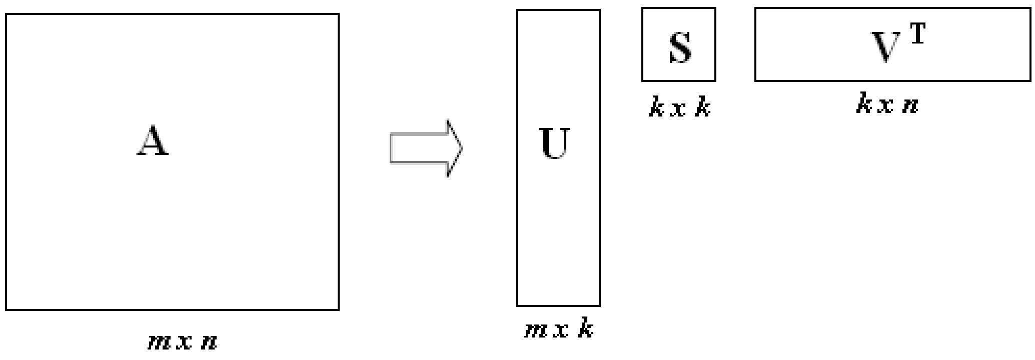


Figure 4.1 - Dimensionality Reduction Process in SVD

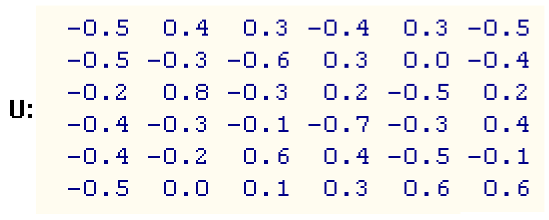
Some researchers [1, 2] claim that the low-rank approximation of the original matrix is better than the original matrix itself. According to them, filtering out of the small singular values can be introduced as removing “noise” data in the matrix. Each customer and product is represented by its corresponding Eigenvector in SVD-based recommender systems. For instance, for a movie recommender system, users who rated similar products are mapped into the space spanned by the same Eigenvectors.

As an example to dimension reduction, consider the <user, movie> rating matrix in Table 4.1.

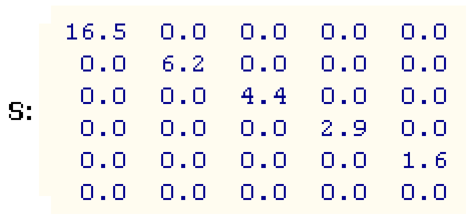
Table 4.1 - Example <User, Movie> Rating Matrix

User/Movie	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
User 1	1	5	0	5	4
User 2	5	4	4	3	2
User 3	0	4	0	0	5
User 4	4	4	1	4	0
User 5	0	4	3	5	0
User 6	2	4	3	5	3

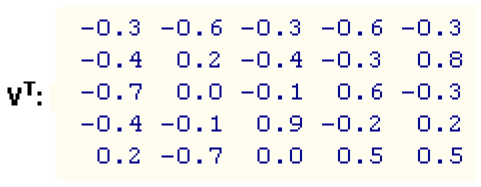
Applying SVD to this matrix:



```
U:  -0.5  0.4  0.3 -0.4  0.3 -0.5
    -0.5 -0.3 -0.6  0.3  0.0 -0.4
    -0.2  0.8 -0.3  0.2 -0.5  0.2
    -0.4 -0.3 -0.1 -0.7 -0.3  0.4
    -0.4 -0.2  0.6  0.4 -0.5 -0.1
    -0.5  0.0  0.1  0.3  0.6  0.6
```



```
S:  16.5  0.0  0.0  0.0  0.0
     0.0  6.2  0.0  0.0  0.0
     0.0  0.0  4.4  0.0  0.0
     0.0  0.0  0.0  2.9  0.0
     0.0  0.0  0.0  0.0  1.6
     0.0  0.0  0.0  0.0  0.0
```



```
VT: -0.3 -0.6 -0.3 -0.6 -0.3
    -0.4  0.2 -0.4 -0.3  0.8
    -0.7  0.0 -0.1  0.6 -0.3
    -0.4 -0.1  0.9 -0.2  0.2
     0.2 -0.7  0.0  0.5  0.5
```

Matrix U (6x6), matrix S (6x5), and matrix V (5x5) are calculated. Now, we will collapse this matrix from a (6x5) space into a 2-Dimensional one. To do this, we simply take the first two columns of U, S and V. The end result:

$$\mathbf{U}: \begin{matrix} -0.5 & 0.4 \\ -0.5 & -0.3 \\ -0.2 & 0.8 \\ -0.4 & -0.3 \\ -0.4 & -0.2 \\ -0.5 & 0.0 \end{matrix} \qquad \mathbf{S}: \begin{matrix} 16.5 & 0.0 \\ 0.0 & 6.2 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{matrix} \qquad \mathbf{V}^{\mathbf{T}}: \begin{matrix} -0.3 & -0.6 \\ -0.4 & 0.2 \\ -0.7 & 0.0 \\ -0.4 & -0.1 \\ 0.2 & -0.7 \end{matrix}$$

For a recommender system based on SVD, here is one very simple strategy: find the most similar user using the 2-Dimensional matrixes above with one of the similarity calculation algorithms and compare his/her items against that of the new user; take the items that the similar user has rated and the new user has not and return them for the new user. Similar to this, for a new item, find the most similar item using the 2-Dimensional matrixes above with one of the similarity calculation algorithms and compare the users rated similar item against the new item; take the users that rate similar item but not the new item and return the ratings for the new item.

4.1.2 SVD Recommendation Algorithms

4.1.2.1 User-based Similarity

In SVD Recommendation, in order to decide whether two users are similar, the reduced matrix $\mathbf{U_k}$ is used [1]. In the ***(mxk)*** $\mathbf{U_k}$ matrix, each row represents a user. The more two rows are similar, the more the users are similar to each other. One critical step in the SVD algorithm is to compute the similarity between users and then to select the most similar users. There are a number of different ways to compute the similarity between users. Here are two such methods: cosine-based similarity and comparing the Euclidian distances in k-dimensional space.

- Cosine-based similarity:

The similarity of two row vectors could be calculated with cosine similarity in order to decide whether two users are similar or not. Formally, similarity between users i and j is stated in (2).

$$sim(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2} \tag{2}$$

Where ``·" denotes the dot-product of the two vectors.

- Comparing the Euclidian distances in k-dimensional space:

For example, in Figure 4.2 the distribution of users in 2D space is demonstrated.

For k=2 →

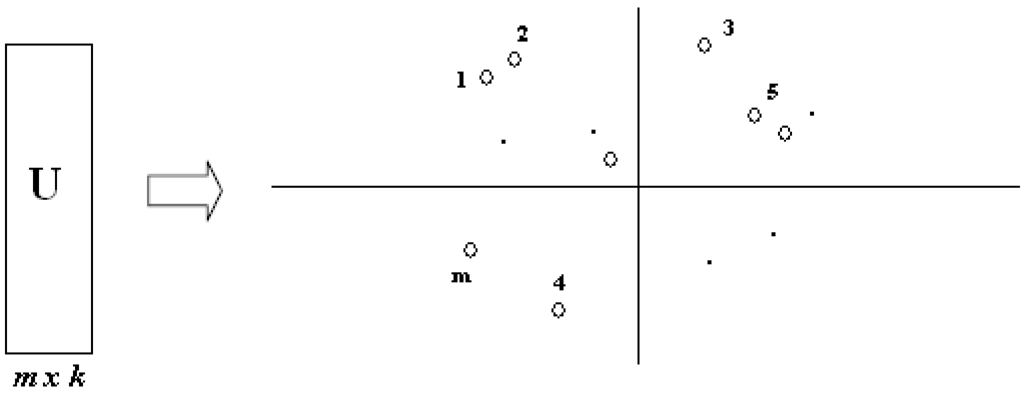


Figure 4.2 - User Sampling in 2D Space for k = 2

To find the most similar user to the user 1, the distances between point number 1 to all points should be calculated and the user which has the smallest distance to the point 1 is the most similar user to user 1.

<u>The algorithm for SVD recommendation by User-based similarity:</u>

Recommendation Request (User = x, Item = y, Rating =?)

1. Find users who rated Item = y from the original matrix A.
2. Find most similar user to User = x among the users who rated Item = y using the reduced matrix $\mathbf{U_k}$.
3. Get the rating of the most similar user to Item = y from the original matrix A and give it for the User = x , Item = y.

For the second part of the algorithm, if the User = x is an already existing user, it exists in the reduced matrix $\mathbf{U_k}$ as a row. If the User = x is a new user, before starting similarity checks, the user has to be projected from n dimensions to k dimensions. Let the ratings of the new user vector is $\mathbf{N_u}$ ***(1xn)***. The projection **P** to the reduced matrix $\mathbf{U_k}$ is made by the formula [1]:

$$\mathbf{P = N_u \times V_k \times S_k^{-1}}$$

In the experimental part of this study, the Euclidian distance algorithm explained above is used for the similarity check of the users. The bottleneck in this technique is the search for similar users among a large user population.

4.1.2.2 Item-based Similarity

Item-based algorithms avoid the bottleneck mentioned in the previous section because they explore item similarities first rather than user similarities. In most systems items are more static than users and not changed very frequently. Therefore, Item-based similarity is suitable for pre-computation. Run time performance of item-based similarity method is better than user-based similarity method [38].

In SVD Recommendation, in order to decide whether two items are similar, the reduced matrix $\mathbf{V_k}$ is used [1]. In the ***(kxn)*** $\mathbf{V_k}$ matrix, each column represents an item. The more two columns are similar, the more the items are similar to each other. In order to compute item-item similarities, similar methods with the user-user similarity, such as cosine-based similarity and comparing Euclidian distances in k-dimensional space, may be used.

The algorithm for SVD recommendation by Item-based similarity:

Recommendation Request (User = x, Item = y, Rating = ?)

1. Find items which are rated by User = x from the original matrix A.
2. Find most similar item to Item = y among the items which are rated by User = x using the reduced matrix $\mathbf{V_k^T}$.
3. Get the rating of the most similar item given by User = x from the original matrix A and give it for the User = x , Item = y.

Similar to user-based similarity, for the second part of the algorithm, if the Item = y is an already existing item, it exists in the reduced matrix $\mathbf{V_k^T}$ as a column. If the Item = y is a new item, before starting similarity checks, first the item has to be projected from n dimensions to k dimensions. Let the ratings of the new item vector is $\mathbf{N_i}$ ***(mx1)***. The projection **P** to the reduced matrix $\mathbf{V_k^T}$ is made by the formula [1]:

$$\mathbf{P = N_i^T \ x \ U_k \ x \ S_k^{-1}}$$

In the experimental part of this study, the Euclidian distance algorithm explained above is used for the similarity check of the items.

4.2 Incremental SVD

In a recommender system, the entire algorithm works in two separate steps. The first step is the offline step and the second step is the online execution step. The user-user and item-item similarity computation steps described above can be seen as the offline step of a recommender system. However, the actual prediction generation is done in run-time in the online step. Usually, the offline computations are very time consuming and is computed infrequently. For instance, a movie recommender site may compute the user-user or item-item similarity tables only once a day or even once a week. If the ratings database is static and if the user behavior does not change significantly over a short period of time, this method works well. Researchers have demonstrated that the SVD-based algorithms can make the similarity formation process of recommender systems highly scalable while producing better results in most of the cases [3, 4, 5]. However the offline step of SVD recommendation, which consists of decomposition and similarity computation steps is computationally very expensive. For an ***(m×n)*** user-item matrix, the SVD decomposition requires a run-time of ***O*(*m*^{3})** [1, 2]. The focus of incremental SVD is to develop an algorithm that ensures highly scalable overall performance. It makes the offline model building of SVD more scalable while achieving prediction quality comparable to the original SVD.

The projection technique is known as "folding-in" in SVD literature [1, 2]. To fold-in new users into the matrix of already reduced user-item matrix $\mathbf{A_k}$, we compute the coordinates for that vector in the basis $\mathbf{U_k}$. Let the size of the new user vector $\mathbf{N_u}$ be ***(1xn)***. The first step in folding-in is to compute a projection **P** that projects $\mathbf{N_u}$ onto the matrix. Such a projection **P** of $\mathbf{N_u}$ is computed as:

$$\mathbf{P = N_u \ x \ V_k \ x \ S_k^{-1}}$$

This user set is then folded-in by appending the k dimensional vector $\mathbf{U_k}$**.** Figure 4.3 [1] below shows a schematic diagram of this process.

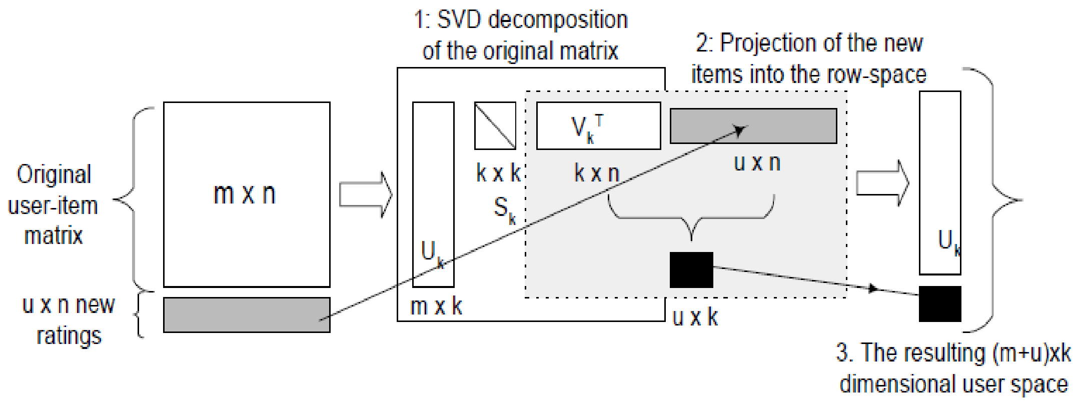

1: SVD decomposition of the original matrix
2: Projection of the new items into the row-space
Original user-item matrix
m x n
u x n new ratings
V_k^T
k x k
S_k
k x n
u x n
U_k
m x k
u x k
U_k
3. The resulting (m+u)xk dimensional user space

Figure 4.3- Semantics of Incremental SVD

To fold-in new items into the matrix of already reduced user-item matrix $\mathbf{A_k}$, we compute the coordinates for that vector in the basis $\mathbf{V_k}$. Let the size of the new item vector $\mathbf{N_i}$ be ***(mx1)***. The first step in folding-in is to compute a projection **P** that projects $\mathbf{N_i}$ onto the matrix. Such a projection **P** of $\mathbf{N_i}$ is computed as:

$$\mathbf{P = N_i^T \times U_k \times S_k^{-1}}$$

This item set is then folded-in by appending the k dimensional vector $\mathbf{V_k}$.

Folding-in is based on the existing model $\mathbf{U_k}$, $\mathbf{S_k}$, and $\mathbf{V_k}$. New users or items do not affect existing user and items. In practice, it is possible to pre-compute the SVD decomposition using ***m*** existing users. For a user-item ratings matrix **A**, the three decomposed matrix $\mathbf{U_k}$, $\mathbf{S_k}$, and $\mathbf{V_k}$ are computed at first. However, when a new set of ratings is added to the database, it is not necessary to compute again the low-dimensional model from the scratch. The folding-in technique provides an incremental system that has the potential to be highly scalable. The complexity of adding a new user or a new item is just O(1) in incremental SVD.

4.3 Contributions to SVD-based Recommendation

4.3.1 Categorization of Users and Items

As it is mentioned before, a recommender system has to provide two properties for its users. One of them is making good accurate recommendations and the other one is fast responding of requests. In order to make more accurate recommendations, we need to make more accurate similarity tables. We propose a new approach that is categorization of items and users to SVD-based recommendation. By this way, we perform SVD-based approach operations on relatively smaller matrices. These smaller matrices are consisting of items in the same category and users in the same category. By this way, we aim to increase the accuracy performance of SVD-based recommendation. Since SVD computational complexity is very high which is $O(m^3)$, smaller matrices also improve the execution time performance of the algorithm.

For instance, for a movie recommender system, the movies can be categorized by their types (action, sci-fi, horror, etc.) and the users can be categorized by their gender, age group, etc. In the Figure 4.5 below, the categorization step of the dataset is demonstrated.

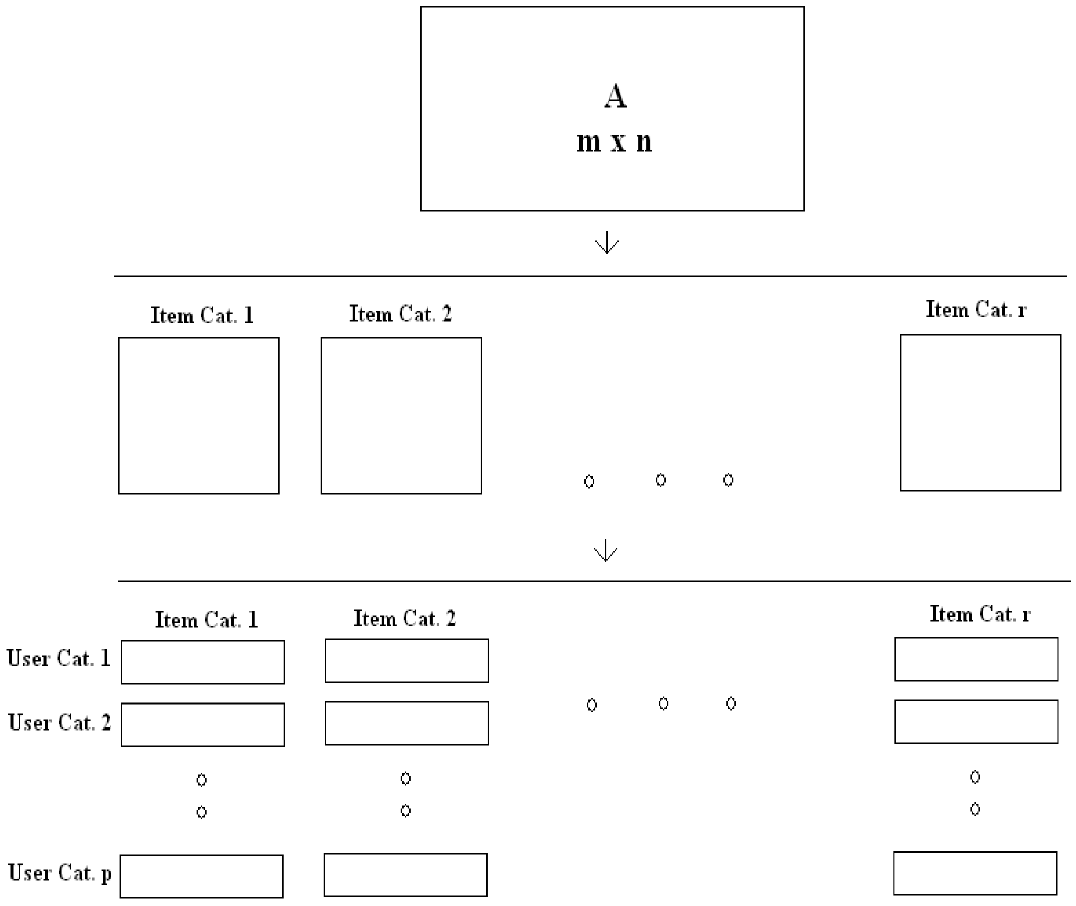


Figure 4.4 - Categorizations Steps of Users and Items

We produce *r x p* matrices from the original matrix. For all of them, the SVD operation is performed separately.

Algorithm for SVD recommendation by categorizing the data (User-User Similarity):

Recommendation Request (User = x, Item = y, Rating = ?)

1. Find the user category for User = x
2. Find the item category for Item = y
3. Find matrix for these categories.
4. Find users who rated Item = y, in this user category.
5. Performing SVD operations on this matrix, find most similar user to User = x among these users.
6. Give the rating of the most similar user to the rating of User = x, Item = y.

Algorithm for SVD recommendation by categorizing the data (Item-Item Similarity):

Recommendation Request (User = x, Item = y, Rating = ?)

1. Find the user category for User = x
2. Find the item category for Item = y
3. Find matrix for these categories.
4. Find items that are rated by User = x, in this item category.
5. Performing SVD operations on this matrix, find most similar item to Item = y among these items.
6. Give the rating of the most similar item to the rating of User = x, Item = y.

In Chapter 5, the experimental results are given for this method.

4.3.2 Adopting Tags to SVD Recommendation

In order to improve recommendation quality, metadata such as content information of items has been used as additional knowledge. With the increasing popularity of the systems which allow users to write tags for items, tags become interesting and useful information to recommendation algorithms. Tags are different from the attributes of users and items. Attributes are global descriptions of items or users, whereas tags are descriptions of items given by the users. In this study, we have tried to use tags for SVD recommendation algorithms and analyzed the results. To adopt tags to normal SVD algorithm, we reduced the three-dimensional matrix<user, item,

tag> to three two-dimensional matrices as <user, item>, <user, tag> and <item, tag>. From these there two-dimensional matrices we construct a new matrix to perform SVD recommendation using explained algorithms in the previous sections.

4.3.2.1 Extension with Tags

Unlike attributes which only have a two-dimensional relation < item, attribute >, tags hold a three-dimensional relation < user, item, tag >. We overcome this three dimensionality problem by projecting it as three two-dimensional problem, < user, tag > and < item, tag > and < user, item > [39].

This can be done by augmenting the standard user-item matrix horizontally and vertically with user and item tags correspondingly. User tags are tags that user u writes to tag items and are viewed as items in the user-item matrix. Item tags are tags that describe an item i by users and play the role of users in the user-item matrix. Figure 4.6 [39] below illustrates this process:

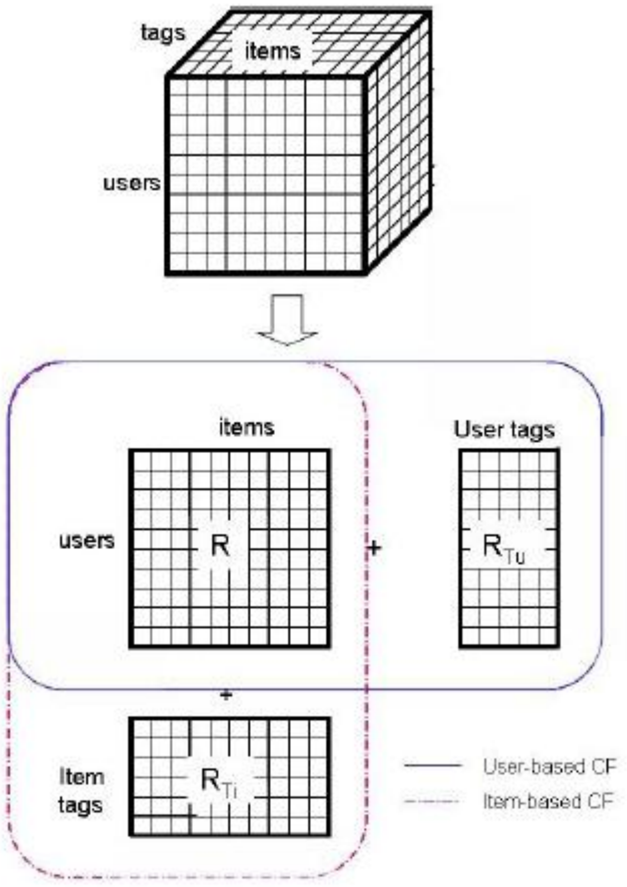


Figure 4.5 - Conversion from 3D Matrix to 2D Matrixes

Moreover, instead of viewing each single tag as user or item, clustering methods can be applied to the tags such that similar tags are grouped together. In this study, we tried substring, java string comparison and edit distance calculation methods to group similar tags in order to improve the performance of recommendation.

After constructing the new two dimensional matrixes, the SVD recommendation algorithms proposed in the previous sections are performed to measure the performance quality improvements. In Chapter 5, the experimental results are given for this method.

Edit Distance

In information theory and computer science, the edit distance between two strings of characters is the number of operations required to transform one of them into the other. There are several different ways to define an edit distance, and there are algorithms to calculate its value under various definitions. We used “Levenshtein Distance” algorithm[41] in this thesis study.

For example, the Levenshtein distance between "kitten" and "sitting" is 3, since the following three edits transform one into the other, and there is no way to do it with fewer than three edits:

1. kitten → sitten (substitution of 's' for 'k')
2. sitten → sittin (substitution of 'i' for 'e')
3. sittin → sitting (insert 'g' at the end).

CHAPTER 5

EXPERIMENTAL RESULTS

The algorithms in the experiments are implemented in java programming language. For SVD and all other matrix calculations, JAMA matrix library [44] is used.

5.1 Dataset

Our experimental dataset is a popular database, the MovieLens dataset by the GroupLens Research group at University of Minnesota [42]. The data set contains 100.000 ratings (1-5 scales) from 943 users on 1682 movies. Each user has rated at least 20 movies.

Ratings are given in the format "user id | item id | rating | timestamp" where user and movie id fields are started from 1 and the time stamps are UNIX seconds since 1/1/1970 UTC.

Information about the movies is given as a tab separated list of:

Movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |

The last 19 fields are the genres which are going to be the movie categories in our experiments.

Information about the users is given as a tab separated list of:

User id | age | gender | occupation | zip code

To compare algorithms, we conducted the experiments on test data which are randomly 80% / 20% splits of the ratings data. We use 5 disjoint test sets for 5 fold cross validation. 5 fold cross validation means, we repeat our experiment with each five training and test set and average the results.

In order to examine and compare the performance of the methods in the experiment, we adopted to mean absolute error (MAE) [43]. The MAE is computed by first summing the absolute errors of the N ratings prediction and then averaging the sum. A smaller value of MAE indicates a better performance.

$$MAE = \frac{\sum_{i=1}^{N} |r_i - \hat{r}_i|}{N}$$

5.2 Results for SVD-based Recommendation by User-User Similarity

SVD-based recommendation by User-User similarity method is implemented as the given algorithm in Chapter 4. The results for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in the Figure 5.1 below:

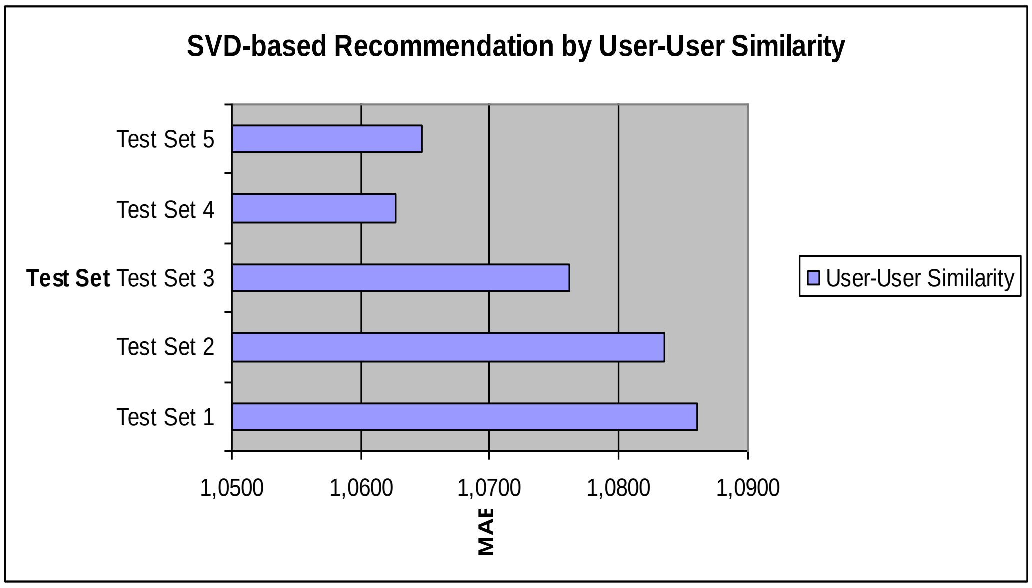


Figure 5.1 - Results of SVD Recommendation by U-U Similarity

The average MAE for 5 disjoint test cases = 1,0746.

5.3 Results for SVD-based Recommendation by Item-Item Similarity

SVD-based recommendation by Item-Item similarity method is implemented as the given algorithm in section 2. The results for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in the Figure 5.2 below:

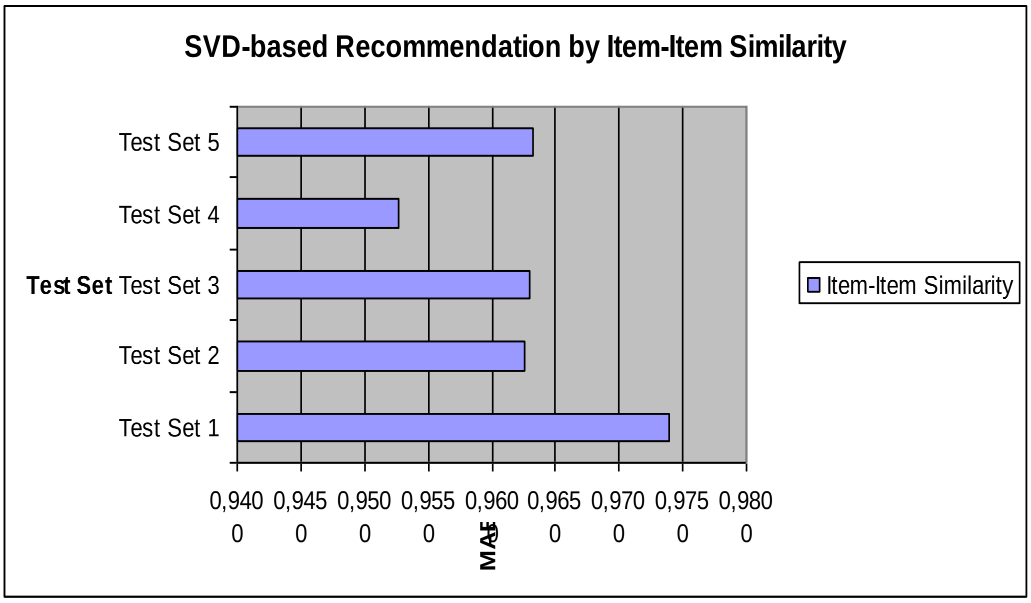


Figure 5.2 - Results of SVD Recommendation by I-I Similarity

The average MAE for 5 disjoint test cases = 0,9631.

From the results for both User-based and Item-based recommendations in Figure 5.1 and 5.2, it is clearly seen that item-based similarity produced better predictions than user-based similarity. Since the number of items is greater than the number of users in the dataset, SVD gives better results for item-item similarity.

5.4 Comparison of Incremental SVD and Normal SVD

According to the incremental SVD algorithm explained in Chapter 4, we add users incrementally to a base set for singular value decomposition.

The prediction accuracy results of incremental SVD and normal SVD methods for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in the Figure 5.3 below:

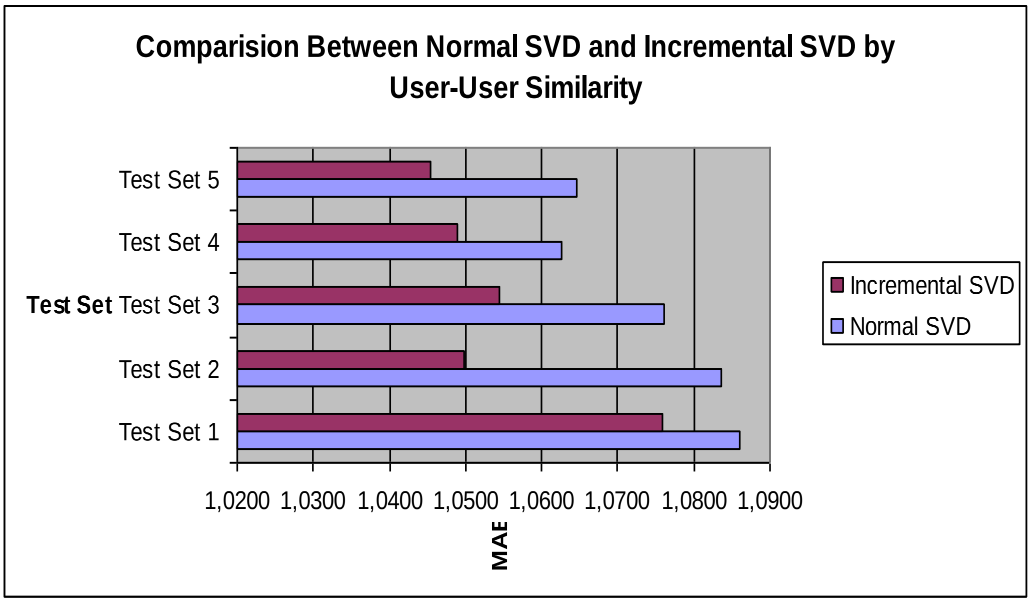


Figure 5.3 - Comparison of SVD and Incremental SVD (U-U)

The average MAE for 5 disjoint test cases for Incremental SVD = 1,0549.

The average MAE for 5 disjoint test cases for Normal SVD = 1,0746.

Moreover, we also examine item-item similarity with incremental SVD by adding movies incrementally to a base set for singular value decomposition.

The prediction accuracy results of incremental SVD and normal SVD methods for item-item similarity for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in Figure 5.4 below:

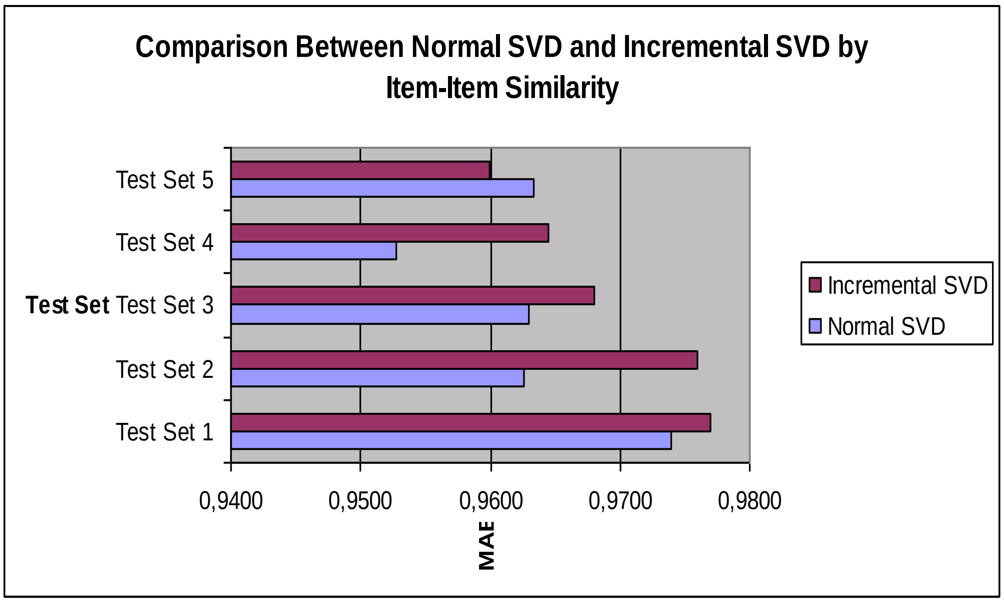


Figure 5.4 - Comparison of SVD and Incremental SVD (I-I)

The average MAE for 5 disjoint test cases for Incremental SVD = 0,9691.

The average MAE for 5 disjoint test cases for Normal SVD = 0,9631.

The results illustrates that, for both user-based and item-based recommendation, normal SVD and incremental SVD have very close prediction performance. On the other hand, while normal SVD operation of the 943 x 1682 <user, movie> matrix takes more than 5 minutes, by incremental SVD it takes approximately 10 seconds.

These results show that, incremental SVD preserves the quality of the predictions compared to normal SVD, whereas it increases the execution time performance significantly. This means incremental SVD overcomes scalability problem to some extent.

5.5 Categorization of Items

According to the movie type, we divided the dataset into categories. In our data set, there are 18 different movie types. These are: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, and Western.

The prediction accuracy results of SVD with categorization and normal SVD methods for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in the Figure 5.5 and 5.6 below:

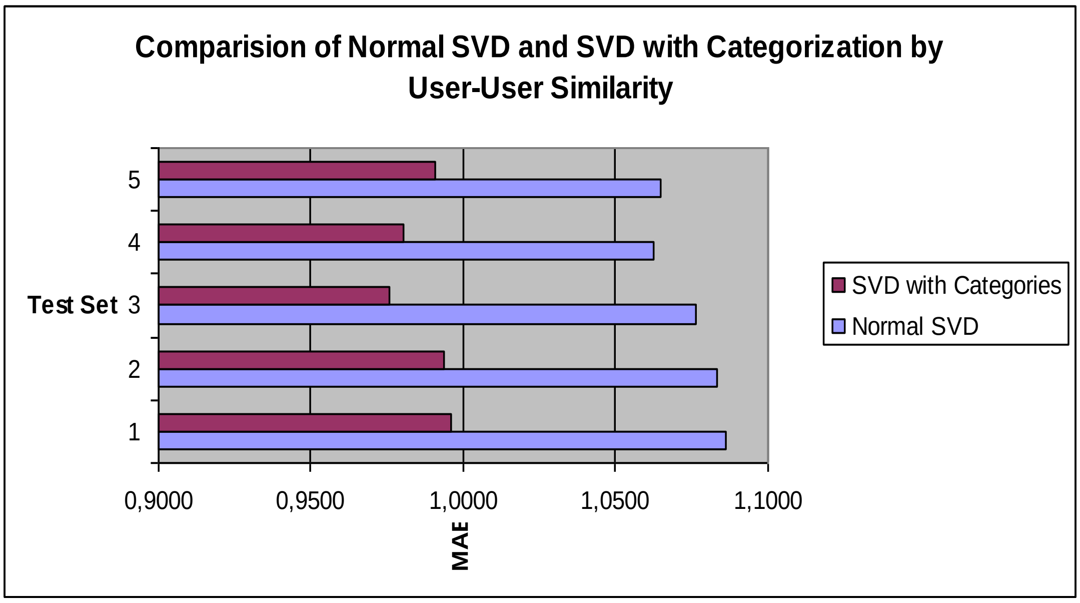


Figure 5.5 - Comparison of SVD and SVD with Categorization (U-U)

MAE for 5 disjoint test cases for Normal SVD by U-U Similarity = 1,0746.

MAE for 5 disjoint test cases for SVD with Categories by U-U Similarity = 0,9874.

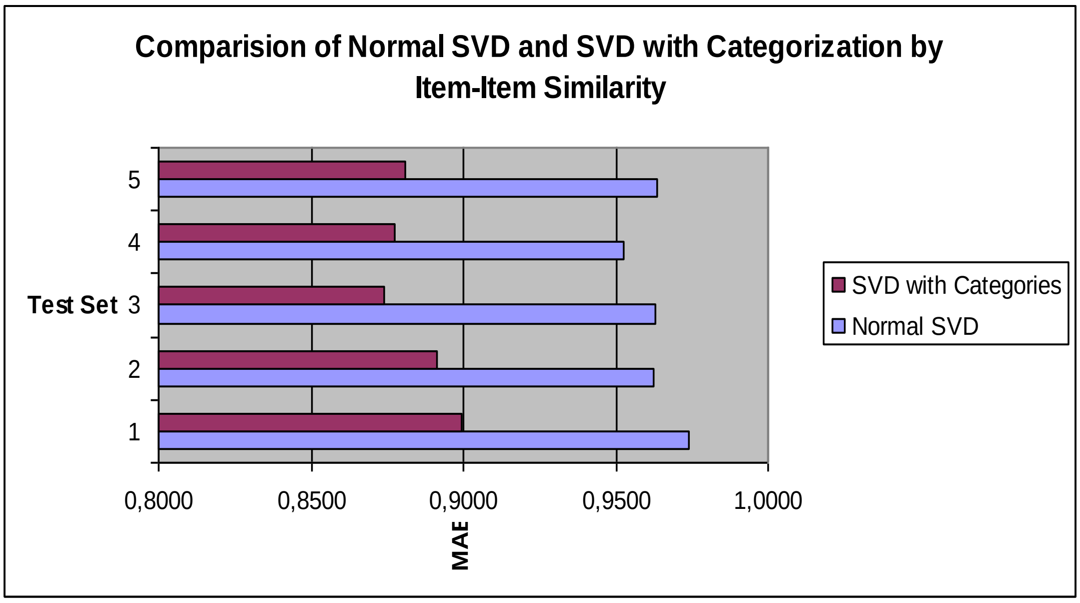


Figure 5.6 - Comparison of SVD and SVD with Categorization (I-I)

MAE for 5 disjoint test cases for Normal SVD by I-I Similarity = 0,9631.

MAE for 5 disjoint test cases for SVD with Categories by I-I Similarity = 0,8846.

For both user-based and item-based recommendation algorithms, after categorizing the movies, the quality of predictions is significantly improved. Moreover, since the computational complexity of SVD operation is $O(m^3)$ and by categorizing the movies, the matrixes become smaller, the computation time is also significantly improved. While normal SVD operation of the 943 x 1682 <user, movie> matrix takes more than 5 minutes, by categorization, due to smaller matrices, SVD takes less than 10 seconds.

5.6 Categorization of Users

According to the gender, we divided the dataset into categories. Users are divided into two groups as Males and Females.

The prediction accuracy results of SVD with categorization and normal SVD methods for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in the Figure 5.7 below:

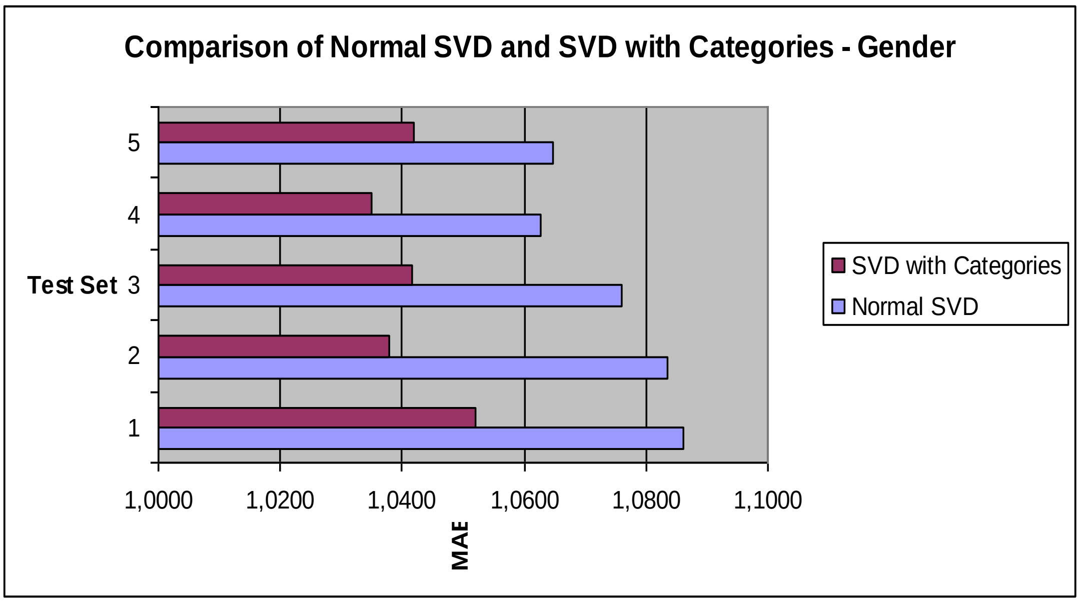


Figure 5.7 - Comparison of SVD and SVD with Categorization (Gender)

MAE for Normal SVD by U-U Similarity = 1,0746.

MAE for SVD with User Categories(Gender) by U-U Similarity = 1,0416.

As a second experiment for User Categories, according to the age group, we divided the dataset into categories. Users are divided into for groups as 0-20, 20-40, 40-60, 60-80.

The prediction accuracy results of SVD with categorization and normal SVD methods for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in the Figure 5.8 below:

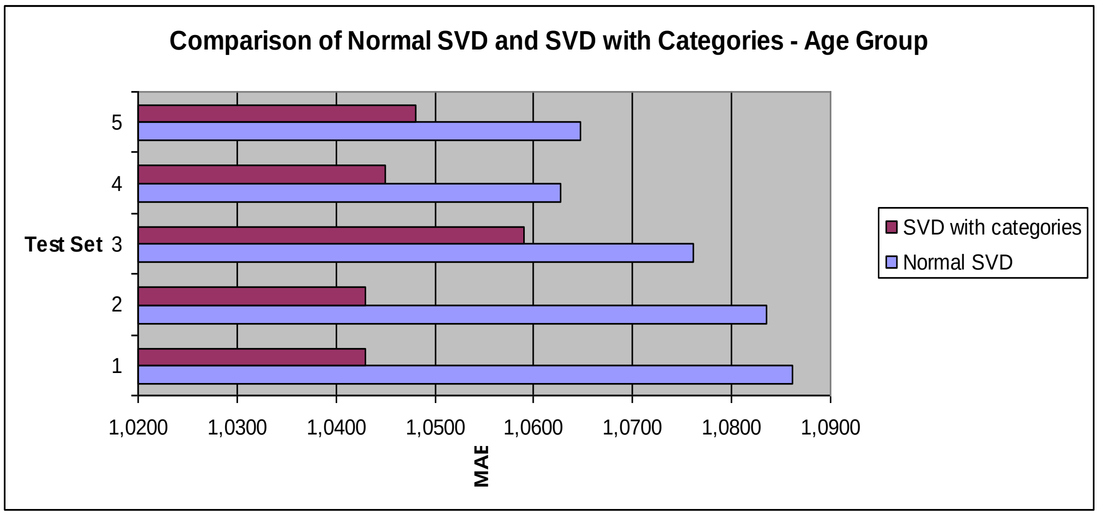


Figure 5.8 - Comparison of SVD and SVD with Categorization (Age)

MAE for Normal SVD by U-U Similarity = 1,0746.

MAE for SVD with User Categories (Gender) by U-U Similarity = 1,0476.

Figure 5.7 and 5.8 demonstrates that, both gender and age group improves the performance of recommendation. Also, computational time decreases by working with smaller matrixes. When we compare the results of categorization of users and categorization of items, it is clearly seen that categorization of items produce better results. However, this is mostly related with the number of categories. For movie categories there exists 19 categories, whereas for user categories there exists 2(gender) and 4(Age Group) categories.

5.7 Categorization of Both Users and Items

According to the user genders and movie types, we divided the dataset into categories. Users are divided into two groups as Males and Females. Movies are divided into 18 categories as Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, and Western.

The prediction accuracy results of SVD with categorization and normal SVD methods for the 5 disjoint test sets, without making any heuristic improvements, are demonstrated in the Figure 5.9 below:

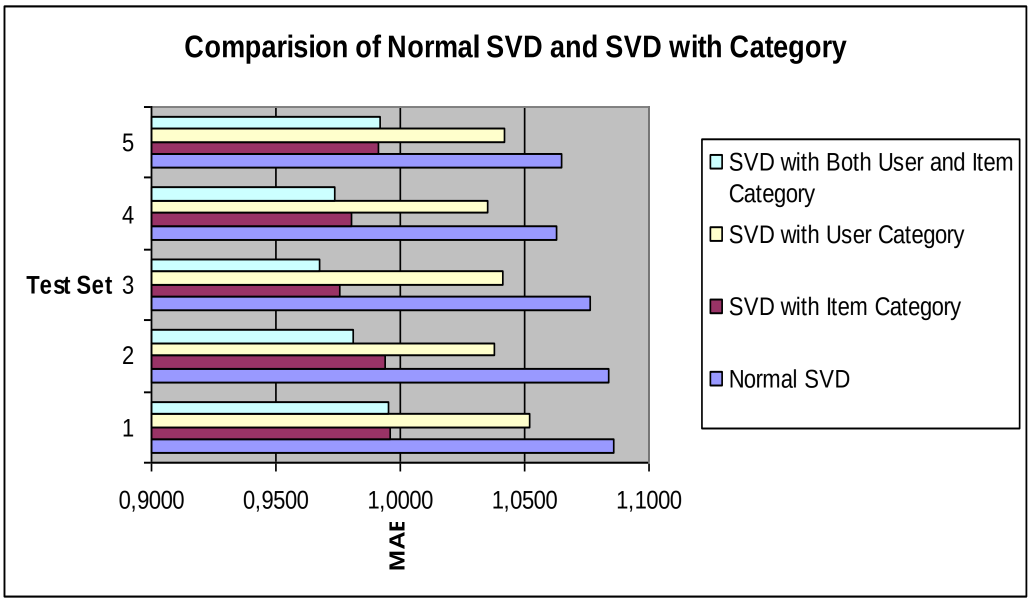


Figure 5.9 - Comparison of SVD and SVD with Categorization (Overall)

MAE for Normal SVD = 1,0746.

MAE for SVD with User Categories = 1,0476.

MAE for SVD with Item Categories = 0,9874.

MAE for SVD with User and Item Categories = 0,9820.

More similar users and items come together to form smaller matrixes by categorization and this increases the performance of recommendation process in two different perspective. The first one is the prediction performance. Figure 5.9 summarizes the contribution of categorization to the prediction quality. And the second one is the computational time performance. Form these views; categorization proposes solutions to sparsity and scalability problems of traditional CF algorithms.

5.8 Adopting Tags to SVD Recommendation

In order to test tag-based SVD recommendation we used a different dataset from the MovieLens by the GroupLens Research group at University of Minnesota [42]. This data set contains 10.000.054 ratings and 95.580 tags applied to 10.681 movies by 71.567 users.

Users were selected at random and all users selected had rated at least 20 movies. Unlike previous data set, no demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in three files, `movies.dat`, `ratings.dat` and `tags.dat`. Movies and ratings are given as similar to the previous data set.

All tags are contained in the file `tags.dat`. Each line of this file represents one tag applied to one movie by one user, and has the following format:

UserID::MovieID::Tag::Timestamp

The lines within this file are ordered first by UserID, then, within user, by MovieID.

Tags are user generated metadata about movies. Each tag is typically a single word, or short phrase. The meaning, value and purpose of a particular tag is determined by each user.

<u>Example for tags:</u>

25::50::Kevin Spacey::1166101426

31::65::buddy comedy::1188263759

31::546::strangely compelling::1188263674

39::277::classic::1188263791

39::1249::woman::1188263764

109::36::death penalty::1211433235

109::1258::Stephen King::1165555223

127::1343::stalker::1188265347

146::10::franchise::1196517851

146::16::imdb top 250::1213424434

146::26::based on a play::1206782429

146::28::based on a book::1210845801

146::32::biology::1208571447

From this we produce a sample dataset consists of randomly selected 1500 users and 1500 movies and we run the tests on this sample dataset. Here are the results in Figure 5.10:

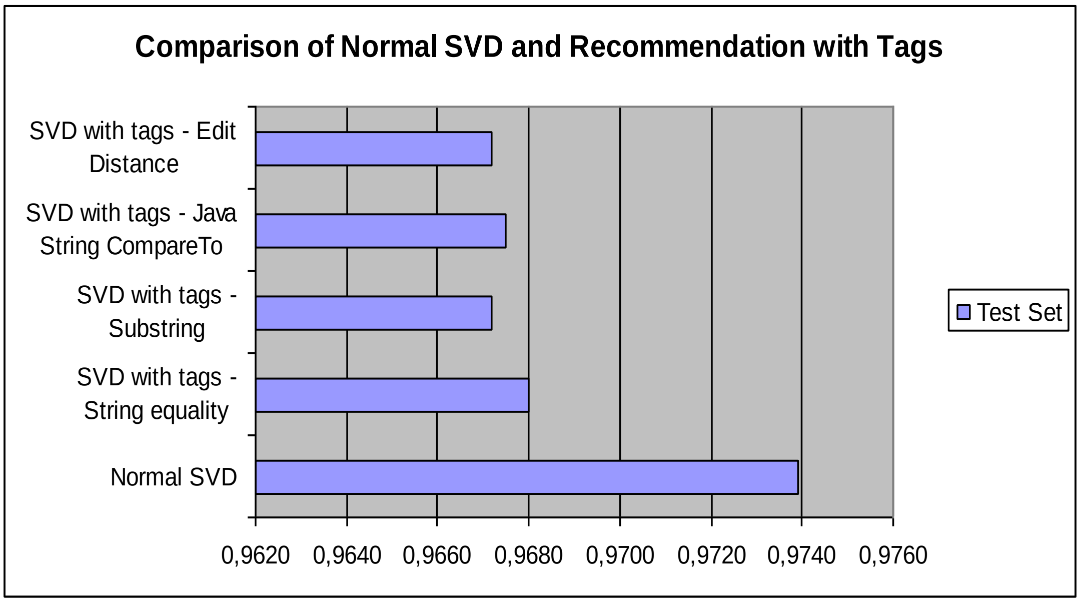


Figure 5.10 - Comparison of SVD and SVD with Tags

MAE for Normal SVD = 0,9739.

MAE for SVD with Tags – String Equality = 0,9680.

MAE for SVD with Tags – Substring = 0,9672.

MAE for SVD with Tags – Java String CompareTo = 0,9675.

MAE for SVD with Tags – Edit Distance = 0,9672.

The results show that, injecting tags into 2-Dimensional SVD improves the performance of recommendation. The reason why effect of categorization is more crucial than effect of tags to the overall performance is the structure of tags in our dataset. In the dataset, tags are given in free formatted texts. Therefore, matching similar tags is quite difficult. We tried some different methods to decide tag similarities such as string equality, substring, CompareTo function of java string class, and edit distance algorithm. The results for all of these methods are demonstrated in Figure 5.10.

From the experiments, we observe the contribution of tags into 2-Dimentional SVD method. However, in order to increase the performance effect of tags adaptation to classical SVD recommendation process, it is necessary to analyze tags, especially semantically check to decide whether they are similar or not and find more similarities among them. For the tags, the important point is that, having more similar tags for the <user, item> pairs. The more similar tags means, the more accurate predictions.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Recommender systems are rapidly becoming an important tool especially on the Web. Recommender system developers have encountered some problems which are currently attractive research areas in the data mining and information retrieval topics for the researchers. The first challenge is to improve the accuracy of the recommendations for the customers. Another challenge is to improve the scalability of the recommendation algorithms.

SVD proposes better results than traditional collaborative filtering algorithms most of the time, however, it includes computationally very expensive matrix calculations and this makes SVD-based recommender systems less suitable for large-scale systems. In this thesis study, SVD-based recommendation techniques are compared with experiments and some new approaches are introduced to this technique. The first contribution we have proposed is the categorization of items and users. Our experiments showed that, item and user categorization increases both the recommendation quality and speed performance of the SVD technique. Moreover, we adopted the tags to the traditional 2-Dimensional SVD approach. By this way, we have chance to analyze the effect of third dimension (tags) to the SVD recommendation performance. Our experiments illustrated that, tags also increase the performance to some extent.

From our experiments, we have derived some future works to increase the performance of SVD-based approach:

- Parallelization of SVD

SVD includes very expensive matrix calculations and this makes it not suitable for large-scale systems. In our experiments, we have tried applying SVD to matrices consist of millions of users and thousands of items. It is impossible to make these calculations in reasonable time. As it is mentioned before, the computational complexity of SVD is O (m^3) which is very expensive. Therefore, as a future work, parallel SVD methods shall be implemented in order to increase the scalability of SVD-based approach. By this way, the effect of the contributions that are suggested in this thesis could be analyzed on very big sample datasets.

- Ontology on Tags

Our experiments show that, grouping similar tags directly affects the performance of 3-Dimensional SVD. If the tags written by two different users are similar to each other, the similarity of them calculated by SVD algorithms becomes very high. Therefore, in order to get better results whether two users are similar or not, it is important to group similar tags. In this thesis, we tried substring, edit distance and some similar methods to decide the similarity of tags. However, these methods are not sufficiently efficient. For the future work, we suggest ontology methods to associate different tags with each other. In computer science, ontology is a formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts. By this way, it will be possible to associate some tags which seem different as written letters but similar semantically.

For instance, tags “good” and “excellent” are semantically similar. However, it is not possible to associate them with string comparison algorithms. For such cases, we think ontology techniques shall be used to relate tags. By having semantically grouped tags, SVD-based approach will produce more accurate results.

REFERENCES

[1] Badrul Sarwar, George Karypis, Joseph Konstan, John Reidl: Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems.

[2] Berry, M. W., Dumais, S. T., and O'Brian, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review, 37(4)*.

[3] Gupta, D., and Goldberg, K. (1999). Jester 2.0: A Linear Time Collaborative Filtering Algorithm Applied to Jokes. In *Proc. of the ACM SIGIR '99*.

[4] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Application of Dimensionality Reduction in Recommender System—A Case Study. In *ACM WebKDD'00 (Web-mining for Ecommerce Workshop)*.

[5] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce. In *Proc. of the ACM EC'00 Conference*. Minneapolis, MN, pp. 158-167.

[6] Anne Yun-An Chen and Dennis McLeod, Collaborative Filtering for Information Recommendation Systems

[7] Schafer, J. B., Konstan, J. and Riedl, J.: 1999, 'Recommender Systems in E-Commerce'. In: *EC '99: Proceedings of the First ACM Conference on Electronic Commerce*, Denver, CO, pp. 158-166.

[8] J. Ben Schafer, Nathaniel Good, and Joseph Konstan et. al. Combining collaborative filtering with personal agents for better recommendations. In Proceedings of the 1999 National Conference of the American Association of Artificial Intelligence, pages 439–436, 1999.

[9] Resnick, P., Varian, H., Recommender Systems. Communications of the ACM, 40, 3, (1997), 56-58.

[10] Terveen, L. and Hill, W: 2001, 'Human-Computer Collaboration in Recommender Systems'. In: J. Carroll (ed.): *Human Computer Interaction in the New Millenium*. New York: Addison-Wesley.

[11] Burke R (2002) Hybrid Recommender Systems: Survey and Experiments.

[12] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In Proc. of ACM Conference on Computer Supported Cooperative Work, 1994.

[13] Shardanand, U. and Maes, P.: 1995, ‘Social Information Filtering: Algorithms for Automating "Word of Mouth"’. In: *CHI ’95: Conference Proceedings on Human Factors in Computing Systems*, Denver, CO, pp. 210-217.

[14] Hill, W., Stead, L., Rosenstein, M. and Furnas, G.: 1995, ‘Recommending and evaluating choices in a virtual community of use’. In: *CHI ’95: Conference Proceedings on Human Factors in Computing Systems*, Denver, CO, pp. 194-201.

[15] Breese, J. S., Heckerman, D. and Kadie, C.: 1998, ‘Empirical analysis of predictive algorithms for collaborative filtering’. In: *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 43-52.

[16] Gediminas Adomavicius, and Alexander Tuzhilin, Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, VOL. 17, NO.6, June 2005.

[17] Heng Luo, Changyong Niu, Ruimin Shen, Carsten Ullrich (2008). A collaborative filtering framework based on both local user similarity and global user similarity

[18] ChoiceStream Technology Brief, Review of Personalization Technologies: Collaborative Filtering vs. ChoiceStream’s Attributized Bayesian Choice Modeling.

[19] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen. Scalable collaborative filtering using cluster-based smoothing. In Proc. of SIGIR, 2005

[20] Jennings, A. and Higuchi, H.: 1993, ‘A User Model Neural Network for a Personal News Service.’ *User Modeling and User-Adapted Interaction*, **3**, 1-25.

[21] Foltz, P. W.: 1990, ‘Using Latent Semantic Indexing for Information Filtering’. In: R. B. Allen (ed.): *Proceedings of the Conference on Office Information Systems*, Cambridge, MA, pp. 40-47.

[22] Condliff, M. K., Lewis, D. D., Madigan, D. and Posse, C.: 1999, ‘Bayesian Mixed-Effects Models for Recommender Systems’. *SIGIR ’99 Workshop on Recommender Systems: Algorithms and Evaluation*. Berkeley, CA. <URL: http://www.cs.umbc.edu/~ian/sigir99-rec/papers/condliff_m.ps.gz>. Last access date is 03.07.2010.

[23] Belkin N. J., Croft W. B., “Information filtering and information retrieval: Two sides of the same coin?”, Communications of the ACM 35, 29–39, December 1992.

[24] K, Lang, “Newsweeder: Learning to filter netnews”, Proceedings of the 12th International Conference on Machine Learning, 1995.

[25] Rich, E.: 1979, ‘User Modeling via Stereotypes’. *Cognitive Science* 3, 329-354.

[26] Krulwich, B.: 1997, ‘Lifestyle Finder: Intelligent User Profiling Using Large-Scale Demographic Data’. *Artificial Intelligence Magazine* **18** (2), 37-45.

[27] Guttman, Robert H.: 1998, ‘Merchant Differentiation through Integrative Negotiation in Agent-mediated Electronic Commerce’. Master’s Thesis, School of Architecture and Planning, Program in Media Arts and Sciences, Massachusetts Institute of Technology.

[28] Towle, B. and Quinn, C.: 2000, ‘Knowledge Based Recommender Systems Using Explicit User Models’. In *Knowledge-Based Electronic Markets, Papers from the AAAI Workshop,* AAAI Technical Report WS-00-04. pp. 74- 77. Menlo Park, CA: AAAI Press.

[29] Konstan, J. A., Riedl, J., Borchers, A. and Herlocker, J. L.: 1998, ‘Recommender Systems: A GroupLens Perspective.’ In: *Recommender Systems: Papers from the 1998 Workshop (AAAI Technical Report WS-98-08).*

[30] Strang, G.: 1988, *Linear Algebra and Its Applications*, New York: Harcourt Brace.

[31] Alspector, J., Koicz, A., and Karunanithi, N.: 1997, ‘Feature-based and Clique-based User Models for Movie Selection: A Comparative Study’. *User Modeling and User-Adapted Interaction* **7**, 279-304.

[32] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D. and Sartin, M.: 1999, ‘Combining Content-Based and Collaborative Filters in an Online Newspaper’. *SIGIR ’99 Workshop on Recommender Systems: Algorithms and Evaluation*. Berkeley, CA. <URL: http://www.cs.umbc.edu/~ian/sigir99-rec/papers/claypool_m.ps.gz>. Last access date is 03.07.2010.

[33] Smyth, B. and Cotter, P.: 2000, ‘A Personalized TV Listings Service for the Digital TV Age’. *Knowledge-Based* Systems **13**: 53-59.

[34] Basu, C., Hirsh, H. and Cohen W.: 1998, ‘Recommendation as Classification: Using Social and Content-Based Information in Recommendation’. In: *Proceedings of the 15th National Conference on Artificial Intelligence*, Madison, WI, pp. 714-720.

[35] Mooney, R. J. and Roy, L.: 1999, ‘Content-Based Book Recommending Using Learning for Text Categorization’. *SIGIR ’99 Workshop on Recommender Systems: Algorithms and Evaluation*. Berkeley, CA. <URL: http://www.cs.umbc.edu/~ian/sigir99-rec/papers/mooney_r.ps.gz>. Last access date is 03.07.2010.

[36] Balabanovic, M.:1998, ‘Exploring versus Exploiting when Learning User Models for Text Representation’. *User Modeling and User-Adapted Interaction* 8(1-2), 71-102.

[37] Kirk Baker, Singular Value Decomposition Tutorial, 2005.

[38] Badrul Sarwar, George Karypis, Joseph Konstan, John Reidl: Item-Based Collaborative Filtering Recommendation Algorithms.

[39] Karen H. L., Tso-Sutter, Leandro B. M., and Lars S. T., Tag-aware Recommender Systems by Fusion of Collaborative Filtering Algorithms.

[40] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*. 41(6).

[41] Wikipedia. <URL: http://en.wikipedia.org/wiki/Levenshtein_distance>. Last access date is 03.07.2010.

[42] ‘Movilens Data Set’. *Grouplens Research Group* <URL: http://www.grouplens.org/node/12>. Last access date is 03.07.2010.

[43] Wikipedia. <URL: http://en.wikipedia.org/wiki/Mean_absolute_error>. Last access date is 03.07.2010.

[44] JAMA: A Java Matrix Package. *The MathWorks and the National Institute of Standards and Technology* <URL: http://math.nist.gov/javanumerics/jama/>. Last access date is 03.07.2010.

APPENDIX A

MATRIX TERMINOLOGY

In Appendix A, matrix basics, Eigenvalues and Eigenvectors, are explained with examples. Examples are taken from the Singular Value Decomposition tutorial [37].

Matrices

A matrix is a table of data, like Table A.1, which shows the top scorers of players in a team in last 5 seasons.

Table A.1 - Top-scorers in last 5 seasons

Top-Scorers	**2005**	**2006**	**2007**	**2008**	**2009**
Player 1	10	11	10	11	11
Player 2	5	12	8	10	2
Player 3	0	4	9	6	6
Player 4	0	4	3	5	6

A table consists of rows, the horizontal list of scores corresponding to a player’s name, and columns, the vertical list of numbers corresponding to the scores for a given season. What makes this table a matrix is that it's a rectangular array of numbers.

As a matrix, Table A.1 looks like this:

$$\begin{bmatrix} 10 & 11 & 10 & 11 & 11 \\ 5 & 12 & 8 & 10 & 2 \\ 0 & 4 & 9 & 6 & 6 \\ 0 & 4 & 3 & 5 & 6 \end{bmatrix}$$

The size, or dimensions, of a matrix is given in terms of the number of rows by the number of columns. This makes the matrix above a 4 x 5 matrix.

A little more formally than before, we can denote a matrix like this:

Let m, n are two integers which are greater than or equal to 1. Let a_{ij}, i = 1, 2,…, m, j = 1, 2, …, n be *(m x n)* numbers. An array of numbers is an *(m x n)* matrix and the numbers a_{ij} are elements of A.

$$A = \begin{bmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ . & & . & & . \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ . & & . & & . \\ a_{m1} & \dots & a_{mj} & \dots & a_{mn} \end{bmatrix}$$

The sequence of numbers A(i) = (a_{i1},…, a_{in}) is the i_{th} row of A, and the sequence of numbers A(j) = ($a_{1j,\dots,}$ a_{mj}) is the j_{th} column of A.

Matrix Terminology

Square Matrix

A matrix is said to be square if it has the same number of rows as columns. To designate the size of a square matrix with n rows and columns, it is called n-square. For example, the matrix below is 3-square.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Transpose

The transpose of a matrix is created by converting its rows into columns; that is, row 1 becomes column 1, row 2 becomes column 2, etc. The transpose of a matrix is indicated with a superscripted T, e.g. the transpose of matrix A is A^T . For example, if

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

then its transpose is

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Matrix Multiplication

It is possible to multiply two matrices only when the number of columns in the first matrix and the number of rows in the second matrix are equal. The resulting matrix has as many rows as the first matrix and as many columns as the second matrix. In other words, if A is a *(m x n)* matrix and B is a *(n x s)* matrix, then the product AB is an *(m x s)* matrix.

The coordinates of AB are determined by taking the inner product of each row of A and each column in B.

$$A = \begin{bmatrix} 2 & 1 & 4 \\ 1 & 5 & 2 \end{bmatrix} B = \begin{bmatrix} 3 & 2 \\ -1 & 4 \\ 1 & 2 \end{bmatrix} AB = \begin{bmatrix} 2 & 1 & 4 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ -1 & 4 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 9 & 16 \\ 0 & 26 \end{bmatrix}$$

$$ab_{11} = \begin{bmatrix} 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix} = 2(3) + 1(-1) + 4(1) = 9$$

$$ab_{12} = \begin{bmatrix} 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \end{bmatrix} = 2(4) + 1(4) + 4(2) = 16$$

$$ab_{21} = \begin{bmatrix} 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix} = 1(3) + 5(-1) + 2(1) = 0$$

$$ab_{22} = \begin{bmatrix} 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \end{bmatrix} = 1(2) + 5(4) + 2(2) = 26$$

Identity Matrix

The identity matrix is a square matrix with entries on the diagonal equal to 1 and all other entries equal zero. The n-square identity matrix is denoted variously as I_{nxn}, I_n, or simply I. The identity matrix behaves like the number 1 in ordinary multiplication, which mean AI = A, as the example below shows.

$$A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix} I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} AI = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix}$$

Orthogonal Matrix

A matrix A is orthogonal if $AA^T = A^TA = I$. For example,

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3/5 & -4/5 \\ 0 & 4/5 & 3/5 \end{bmatrix}$$

is orthogonal because

$$A^T A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3/5 & -4/5 \\ 0 & 4/5 & 3/5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3/5 & 4/5 \\ 0 & -4/5 & 3/5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Diagonal Matrix

A diagonal matrix A is a matrix where all the entries a_{ij} are 0 when i = j.

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & & 0 \\ . & & . & . \\ 0 & \dots & & a_{mm} \end{bmatrix}$$

Determinant

A determinant is a function of a square matrix that reduces it to a single number. The determinant of a matrix A is denoted |A| or det(A). If A consists of one element a, then |A| = a; in other words if A = [6] then |A| = 6.

If A is a *(2 x 2)* matrix, then

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

For example, the determinant of

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}$$

is

$$|A| = \begin{vmatrix} 4 & 1 \\ 1 & 2 \end{vmatrix} = 4(2) - 1(1) = 7.$$

Finding the determinant of an n-square matrix for n > 2 can be done by recursively deleting rows and columns to create successively smaller matrices until they are all *(2 x 2)* dimensions, and then applying the previous definition.

For example for a *(3 x 3)* matrix:

$$\begin{vmatrix} -1 & 4 & 3 \\ 2 & 6 & 4 \\ 3 & -2 & 8 \end{vmatrix} = (-1)\begin{vmatrix} 6 & 4 \\ -2 & 8 \end{vmatrix} - (4)\begin{vmatrix} 2 & 4 \\ 3 & 8 \end{vmatrix} + (3)\begin{vmatrix} 2 & 6 \\ 3 & -2 \end{vmatrix} =$$

$$-1(6 \cdot 8 - 4 \cdot -2) - 4(2 \cdot 8 - 4 \cdot 3) + 3(2 \cdot -2 - 3 \cdot 6) =$$

$$-56 - 16 - 66 = -138$$

Eigenvectors and Eigenvalues

An Eigenvector is a nonzero vector that satisfies the equation

$$A\vec{v} = \lambda\vec{v}$$

Where A is a square matrix, λ is a scalar, and v is the Eigenvector. λ is called an Eigenvalue. Eigenvalues and Eigenvectors are also known as, respectively, characteristic roots and characteristic vectors, or latent roots and latent vectors. Eigenvalues and Eigenvectors can be calculated by treating a matrix as a system of linear equations and solving for the values of the variables that make up the components of the Eigenvector. For example, finding the Eigenvalues and corresponding Eigenvectors of the matrix

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

means applying the above formula to get

$$A\vec{v} = \lambda\vec{v} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

in order to solve for λ; x1 and x2. This statement is equivalent to the system of equations

$$2x_1 + x_2 = \lambda x_1$$

$$x_1 + 2x_2 = \lambda x_2$$

which can be rearranged as

$$(2-\lambda)x_1 + x_2 = 0$$
$$x_1 + (2-\lambda)x_2 = 0$$

A necessary and sufficient condition for this system to have a nonzero vector [x1; x2] is that the determinant of the coefficient matrix

$$\begin{bmatrix} (2-\lambda) & 1 \\ 1 & (2-\lambda) \end{bmatrix}$$

is equal to zero. Accordingly,

$$\begin{vmatrix} (2-\lambda) & 1 \\ 1 & (2-\lambda) \end{vmatrix} = 0$$

$$(2-\lambda)(2-\lambda) - 1 \cdot 1 = 0$$

$$\lambda^2 - 4\lambda + 3 = 0$$

$$(\lambda - 3)(\lambda - 1) = 0$$

There are two values of λ that satisfy the last equation; thus there are two Eigenvalues of the original matrix A, and these are $\lambda_1 = 3$; $\lambda_2 = 1$.

Eigenvectors which correspond to these Eigenvalues can be calculated by plugging λ back in to the equations above and solving for x_1 and x_2. To find an Eigenvector corresponding to $\lambda = 3$, start with

$$(2-\lambda)x_1 + x_2 = 0$$

And substitute to get

$$(2-3)x_1 + x_2 = 0$$

Which reduces and rearranges to

$$x_1 = x_2$$

There are an infinite number of values for x_1 which satisfy this equation; the only restriction is that not all the components in an Eigenvector can equal zero. So if $x_1 = 1$, then $x_2 = 1$ and an Eigenvector corresponding to $\lambda = 3$ is [1; 1].

Finding an Eigenvector for $\lambda = 1$ works the same way.

$$(2-1)x_1 + x_2 = 0$$

$$x_1 = -x_2$$

So an Eigenvector for $\lambda = 1$ is [1; -1].