Project Number: MQP-SDO-207

Machine Learning Applications

A Major Qualifying Project Report

Submitted to The Faculty

of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Dennis Silva, Jr.

March, 2016

Approved:

Professor Sarah Olson

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see http://www.wpi.edu/academics/ugradstudies/project-learning.html

Abstract

Machine learning is an interdisciplinary approach in building mathematical models from known inputs to make data-driven predictions and decisions. In this project, we provide a background for the different forms of machine learning and the plethora of scientific applications it is used for. We then narrow our view specifically to item recommendation systems and present the mathematical and statistical methods used to generate predictive recommendations to a system's users. The project culminates in a sample recommendation system we programmed in MATLAB utilizing publicly available movie and joke rating data sets.

Executive Summary

Today we live in an age abundant with data. Due to the increasing use of web and cloud based services, as well as advances in computer and sensor technology, a wealth of new data is being gathered throughout academia, business, and government. With there being such a large influx of data, the term 'big data' has been used as a common term to describe data sets which are so large in volume, are generated very quickly, or are so complex that traditional data processing techniques are inadequate in revealing the underling patterns and trends. In order to handle this big data for use in a variety of applications, many have utilized and built upon the methods and techniques developed in a domain called machine learning. Machine learning focuses on making data-driven predictions and decisions based on known and unknown properties in data collected. Machine learning algorithms take techniques and theorems found in linear algebra, probability theory, statistics, and numerical analysis.

This paper serves to be an introduction into both the applications as well as some of the fundamental mathematical techniques underpinning various machine learning practices. In this paper, the three major machine learning systems, supervised, unsupervised, and reinforcement learning are described. These systems are categorized based on the type of feedback or knowledge the user has access to. The applications of these systems are then discussed from varying perspectives. Applied mathematicians can utilize machine learning to find near-optimal solutions to classical NP-hard problems such as the traveling salesman problem. Game theorists can simulate or model finite evolutionary games using these techniques as well. There have also been several medical and biological applications that can better predict and prognose illnesses such as cancer in individuals. More lucrative applications to machine learning in online advertising are also discussed such as the algorithms behind Google's advertising service AdWords.

Recommendation systems are yet another major application of machine learning, and as such, it is the major application discussed in this paper. Mathematical and statistical methods for data analysis are discussed within the scope of recommendation systems. Principal Component Analysis is one of these methods which serves to help derive a low-dimensional uncorrelated set of features from a large set of correlated variables. Singular Value Decomposition is a matrix factorization method that is used within Principal Component Analysis to help minimize numerical error. In addition, neighborhood formation is described for finding items that are similar to one another based upon some similarity computation technique. The three techniques discussed are the Pearson correlation, cosine-based similarity, and adjusted cosine similarity. To evaluate these systems for recommendation or prediction accuracy, scientists utilize various predictive accuracy metrics. Two rating prediction measures, Mean Absolute Error and Root Mean Squared error are discussed. Three usage prediction measures are included as well: Precision, Recall, and the False Positive Rate.

In order to apply the knowledge above, we coded two recommendation algorithms in MATLAB, ItemSvdRec and ItemSvdDemoRec. These algorithms utilize collaborative and content-based filtering to predict the ratings users would give to certain items if given the opportunity to view or use them. The main data sets tested over, MovieLens 100K and MovieLens 1M, are from the MovieLens online movie recommender system website. We provide and discuss our results from the five predictive accuracy metrics described above. We also provide results over an alternative data set, Jester 2, which has ratings who differ in scale and complexity than the MovieLens data sets.

Acknowledgments

I would like to give many thanks to my advisor Professor Sarah Olson for her guidance and encouragement throughout this project. The data analysis techniques, additional resources, and overall learning structure she provided gave me the opportunity to research an area of mathematics I have now come to immensely enjoy.

Contents

1 Introduction and Motivation **6**
- 1.1 Introduction . . . 6
- 1.2 Machine Learning Systems . . . 6
- 1.3 Machine Learning Applications . . . 10
- 1.4 Key Terms and Notation Review . . . 14

2 Methods for Data Analysis: Recommendation Systems **16**
- 2.1 Principal Component Analysis . . . 16
- 2.2 Singular Value Decomposition . . . 19
- 2.3 Neighborhood Formation . . . 23
- 2.4 Predictive Accuracy Metrics . . . 26

3 Results: MATLAB Models **30**
- 3.1 Coded Algorithm Descriptions . . . 30
- 3.2 MovieLens Data Sets . . . 31
- 3.3 Prediction Evaluation Metric Results . . . 33

4 Discussion **40**
- 4.1 Performance over MovieLens Data Sets . . . 40
- 4.2 Alternative Data Set: Jester Online Joke Recommender . . . 43
- 4.3 Conclusion and Future Research . . . 44

A MATLAB Code **47**
- A.1 Item Filtering with SVD Recommender (ItemSvdRec) . . . 47
- A.2 Item Filtering with SVD and Item Demographics Recommender (ItemSvdDemoRec) . . . 49
- A.3 Data to Matrix: MovieLens 100K Example . . . 51
- A.4 5 Fold Cross Validation Data Sets Creation . . . 52

List of Figures

1.1 Data Never Sleeps 3.0 Infographic . . . 7
1.2 Supervised Learning Flow Chart . . . 9
1.3 Reinforcement Learning Diagram . . . 10
1.4 Growth of GenBank Nucleotide Sequences (1992-2008) . . . 11
1.5 Google Adwords Auction Method . . . 13

2.1 Householder Reduction and Golum-Reinsch Algorithms for SVD . . . 24
2.2 Item Similarity Computation . . . 25
2.3 Item-based Rating Prediction . . . 26

3.1 Sparsity of MovieLens 100K . . . 32
3.2 Ratings Variance by Singular Value . . . 34
3.3 Demographic Variance by Singular Value . . . 34
3.4 MAE of Rating Predictions: MovieLens 100K . . . 35
3.5 MAE of Rating Predictions: MovieLens 1M . . . 36
3.6 RMSE of Rating Predictions: MovieLens 100K . . . 36
3.7 RMSE of Rating Predictions: MovieLens 1M . . . 37
3.8 Precision of Usage Prediction: MovieLens 100K . . . 37
3.9 Precision of Usage Prediction: MovieLens 1M . . . 38
3.10 Recall of Usage Prediction: MovieLens 100K . . . 38
3.11 Recall of Usage Prediction: MovieLens 1M . . . 39

4.1 Ratings Variance by Singular Value: Jester 2 . . . 44
4.2 Rating Prediction Metrics: Jester 2 . . . 45
4.3 Usage Prediction Metrics: Jester 2 . . . 45

List of Tables

2.1 Recommender Usage Classification 28

4.1 Parameter Evaluation Comparison: MovieLens 100K 41
4.2 Optimal Metric Comparisons 43

Chapter 1

Introduction and Motivation

1.1 Introduction

Today we live in an age abundant with data. Due to the increasing use of web and cloud based services, as well as advances in computer and sensor technology, a wealth of new data is being gathered throughout academia, business, and government. Following similar to that of Moore's Law, it is estimated that the total global volume of data collected and stored more than doubles every two years [30]. The internet population today represents over 3.2 billion people who alone generate thousands of petabytes (10^{15} bytes) of data every day [13]. Figure 1.1 is an infographic created by the software company Domo depicting the shear volume of data that is processed every minute of every day online. With there being such a large influx of data, the term 'big data' has been used as a common term to describe data sets which are so large in volume, are generated very quickly, or are so complex that traditional data processing techniques are inadequate in revealing the underling patterns and trends. In order to handle this big data for use in a variety applications, many have utilized and built upon the methods and techniques developed in a subfield of computer science called machine learning. *Machine learning* focuses on the study and construction of algorithms that can make predictions on data through a learning process [23]. All machine learning algorithms take as input a *training set*, a data set that is assumed true, to make data-driven predictions and decisions. How and whether or not the training set is labeled defines what learning system type is used. Although typically classified as a subfield of computer science, machine learning is an interdisciplinary field with strong fundamental ties to pure and applied mathematics. Techniques and theorems found in linear algebra, probability theory, statistics, and numerical analysis help form the basis of many popular machine learning algorithms and applications.

1.2 Machine Learning Systems

The phrase "machine learning" can be described in a great deal of ways. Arthur Samuel, creator of the first self-learning program The Samuel Checkers-playing Program, defines machine learning as "a field of study that gives computers the ability to learn without being explicitly programmed" [41]. Another pioneer in machine learning and artificial intelligence, Tom Mitchell, provides a formal definition of machine learning: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [31]. The terms "machine learning" and "data mining" are often

Figure 1.1: Data Never Sleeps 3.0 is an infographic created by the software company Domo depicting the amount of data processed by popular internet sites and services every minute. It reflects just a small fraction of the data being generated every day and reveals the need to utilize efficient machine learning techniques to analyze it all. Taken from [13].

employed together. While data mining focuses on the discovery of previously unknown properties in data, machine learning focuses on making new predictions based on known properties using accessible training data. Essentially, the goal is to develop learning algorithms that can be programmed to solve new problems using previous examples rather than directly programming algorithms to solve those new problems as they arise. These learning algorithms, or the tasks they solve, are commonly divided into three major categories depending on the type of feedback the learner has access to. These three categories are supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning falls on one end of the machine learning algorithm spectrum. *Supervised learning* is utilized when some set of data is already known to be classifiable, but one needs a classifier or algorithm to actually sort through it. It is a learning system that utilizes labeled training data or training examples. A training example is a data pair consisting of an input object and its corresponding output object known generally as a label. The goal is to build a model of the distribution of class labels in terms of some predictor features [25]. Thus, the model will generate a general rule that can map new inputs with particular set of features to outputs or labels. A flow chart depicting a generalized process of supervised learning is shown in Figure 1.2. For example, take the simple to describe yet complicated task of text classification. One may want to develop a model that can classify handwritten text to be read by a computer. First data samples of handwritten text must be collected to train the system. Pre-processing of the data must be done such as digitizing samples, removing illegible writing, etc. Definitions or labels must be attributed to each data sample by some outside process. A machine learning algorithm is then selected and training begins. The handwritten text samples are given as input to the machine algorithm along with the known classifications of those samples. The algorithm must utilize the characteristics of the text to distinguish between different letters or phrases. Once training is complete, the algorithm is provided a variety of new samples without known classifications. The algorithm should provide a label as output of what it classified the input as based on previous inputs. Evaluation then occurs by comparing the algorithm's guess, in this case output label, to the signal, the actual known label. If a pre-defined amount and variety of the handwriting samples are correctly classified by the algorithm, the system can be called a successful classifier. If not, the system must be modified in some way. Most individuals will turn to parameter tuning first. With most machine learning algorithms, additional parameters are needed outside of the object to be classified. These parameters are directly related to the algorithm being used, and vary from system to system. If tuning does not better classification, a new algorithm may need to be selected or new data may need to be collected.

Unsupervised learning lies at the opposite end of supervised learning on the machine learning spectrum. *Unsupervised learning* attempts to find latent or underlying structures from unlabeled sets of data. Unlike supervised learning, training data is not labeled and at times cannot be classified by some simple external process. Thus, one uses this form of learning when they face the question of how to best classify a set of data. Unsupervised learning follows a similar design process to supervised learning except for a few key steps. Similarly, one must first identify, gather, and pre-process related data, but now without utilizing labels. At times labels can help determine what data needs to be collected, to validate the data's usefulness, or even how to separate the data into a training set. Without labels, these tasks can become more difficult and more care must be given to organizing your data before input into the machine learning algorithm. In addition, due to this lack of output objects or labels, there is no error or reward signal to arrive at a potential solution or classification. Instead, most unsupervised learning algorithms seek to measure hidden similarities between data based upon some type of metric [38]. Most unsupervised learning methods classify inputs through clustering. Data points within a given cluster are said to have a higher measure in similarity, in terms of a defined metric, than data

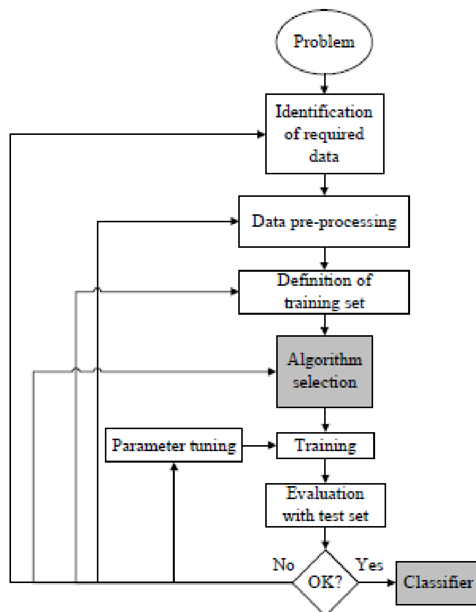


Figure 1.2: A generalized flow chart of supervised learning classification. A problem is first specified, and then data is identified and pre-processed to form a labeled training data set. An algorithm is then selected and uses inputs and their labels to learn or model features of the inputs. Different inputs with similar features are then classified, given labels, by the system to evaluate its accuracy. Taken from [25].

.

in any other cluster. An example of a procedure to aid in the clustering of data, Principal Component Analysis, can be found in Section 2.1. Evaluation of an unsupervised system also differs from its supervised counterpart. One cannot simply evaluate the system through comparing the number of successful classifications of inputs by the system to the total number of inputs provided. Evaluation metrics must reflect the type of data being classified and they tend to vary from system to system. Examples of alternative evaluation metrics related to unsupervised recommendation systems can be found in Section 2.4.

Reinforcement learning falls in the middle of the machine learning spectrum. *Reinforcement learning* is a system similar to that of supervised learning, but differs in that the training data provided is of the form of a scalar reinforcement signal rather than a label. This numerical value constitutes a measure of how well the system operates [42]. The general goal is to find the existence of and characterize optimal solutions given a set of rules and scalar value feedback. Reinforcement learning is best used when there is no simple way to classify the data, but one knows whether it is better or worse than other data. A diagram for the reinforcement learning process can be seen in Figure 1.3. For illustration purposes, suppose one wants to devise a reinforcement learning system to play tic-tac-toe. In this game two players take turns on a three-by-three board. Players take turns placing their mark, X or O, on the board until the board is filled or a player has placed three of their marks in a row on the board. The player who marks an entire row wins. In terms of Figure 1.3, the players are agents and the environment is encompassed by the game rules. At first an agent must choose an action to take without knowledge of whether it is "good" or "bad." Once the action is taken, the state of the environment is changed and the agent receives some numerical, possibly negative, reward associated with that action. This reward is calculated from some pre-defined metric set by the creator of the algorithm. Once the opposing agent makes their move, the first agent must evaluate their environment

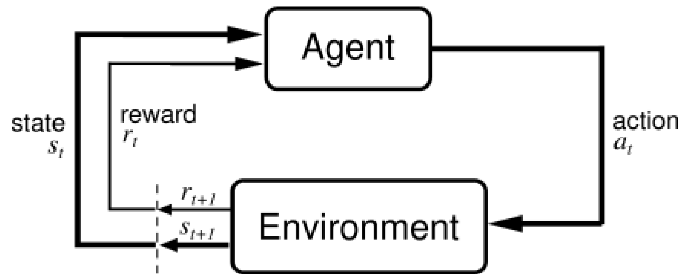


Figure 1.3: The classical representation of a reinforcement learning algorithm. The algorithm serves to map a situation in an environment to an action, and is popular in game theory. An agent or decision maker takes an action. This action changes the state of the environment and the agent receives a, possibly negative, reward associated with that action. The agent accumulates rewards for different actions and will tend to chose actions that increase their overall reward more frequently over several iterations of the algorithm. Reproduced from [42].

using the accumulation of the past rewards it received. Another action is then selected and the process repeats until stopped by the game rules or the creator. In the case of tic-tac-toe and similar games, one may use cumulative rewards from past games to gather more data.

1.3 Machine Learning Applications

Applied mathematics has begun to prosper from machine learning, providing near-optimal or sufficient solutions to complex problems within reasonable time frames. Mathematicians have utilized machine learning to tackle various NP-hard problems that arise in operations research. A nondeterministic polynomial (NP) time problem is one which is solvable in polynomial time by a nondeterministic Turing machine. A NP-hard problem is one such that an algorithm solving it can be translated into another solving any other NP-problem [9]. A classical NP-hard problem that has been investigated through the lens of machine learning is the traveling salesman problem (TSP). In terms of graph theory, TSP is an optimization problem for finding the least-cost cyclic route that passes through all vertices of a weighted graph. TSP, and variations of it, commonly appear in planning, logistics, and vehicle routing problems. Machine learning algorithms have been created and shown to outperform other heuristic techniques in solving certain variations of the TSP. For example, Ant-Q is a family of algorithms shown to perform very well on symmetric versions of the TSP. Symmetric TSP involves graphs that are undirected, or in other words, the weight of the edge connecting vertex A to vertex B is the same as the weight from B to A [14]. Another example includes the creation of a machine learning algorithm to help solve the traveling repairman problem which is similar to TSP, but also includes a time-probabilistic failure rate at each vertex [43]. Game theorists have used similar machine learning algorithms to identify particular game-states and optimal strategies of a particular game. Evolutionary games where agents mutate through each successive generation have been modeled through reinforcement machine learning [32]. The computationally intensive solving for the nash equilibria of some finite turn graph-based games, such as The Territorial Raider Game, can be done for very complex graphs using reinforcement machine learning as well [16].

Machine learning is particularly well-suited for analyzing the hard to detect patterns in large or

noisy data sets found in medical and biological applications. Cancer researchers have been utilizing machine learning techniques such as artificial neural networks and decision trees for cancer detection and diagnosis since the 1980s [40]. Since then, machine learning techniques in cancer research has grown considerably. According to the latest PubMed statistics, as of 2013 there are over 8400 papers published on machine learning methods to identify, classify, detect, or distinguish tumors and other malignancies [10]. Recently machine learning has expanded from primarily focusing on diagnosis and detection of cancer to prediction and prognosis as well. One study formulated machine learning approaches to predict lung cancer in patients based on mass spectrometry data [33]. Another study created a prognostic model using statistical and machine learning techniques that makes quantitative estimates of the probability of relapse for breast cancer patients [11]. A more detailed paper on several applications of machine learning techniques to the detection, diagnosis, and management of cancer can be found in [28]. Another area that has accumulated large sets of data over the years is phylogenetics. Phylogenetics is the study of evolutionary relationships between groups of organisms. Phylogenetic trees or representations of an organisms' evolution were traditionally constructed based on different morphological or metabolic features [26]. Over the past few decades however, there has been an astounding increase in publicly available genome sequences through databases such as GenBank [5]. Figure 1.4 reveals the exponential growth of GenBank's data showing that from 1994 to 2008 GenBank acquired nearly 100 million nucleotide sequences [5]. Due to this abundance in genomic data, phylogenetic trees are now computed through comparing various genomes through a process called multiple sequence alignment. Machine learning optimization techniques have been shown to greatly aid in this process [3].

[H]

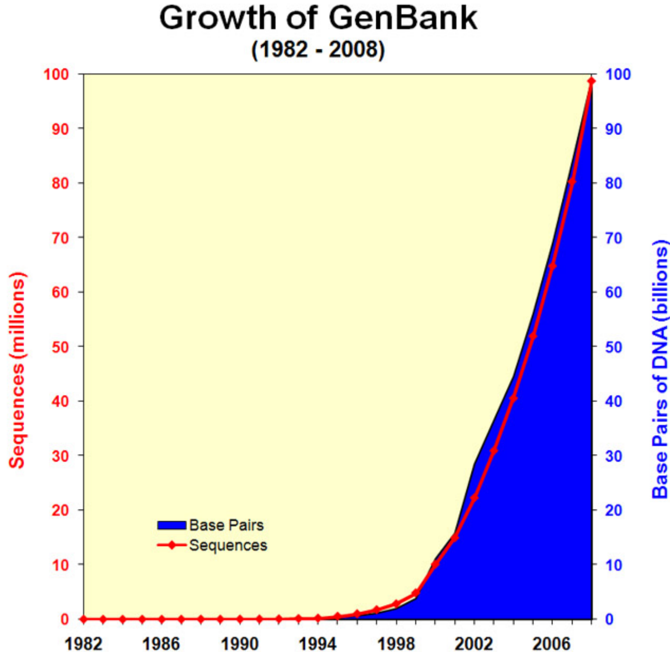


Figure 1.4: The growth of GenBank from 1982 to 2008. GenBank is a comprehensive database that contains publicly available nucleotide sequences for over 260,000 formally described species. The database is used throughout genomics and phlyogenetics research and has provided massive amounts of data for machine learning practices. Taken from [5].

With the massive amount of web pages available to the average user, retrieving accurate and reliable information from the Web has become a primary concern for companies. Of the several different forms of information retrieval done on the Web, page ranking may be one of the most important. Page ranking is a way to assign numerical weights to individual elements of some set of connected documents. In terms of the Web, page ranking allows for a measurement of the relative importance of a web page compared to all other accessible pages. PageRank was one of the first algorithms used by the Google search engine, and it played a large role in Google's success [6]. PageRank utilizes only the link structure of the web to create a ranking. The PageRank of a web page P, $PR(P)$, is given by the recursive equation:

$$PR(P) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)}\right) \tag{1.1}$$

where $PR(T_i)$ is the PageRank of page T_i which links to page P, $C(T_i)$ is the total number of outbound links on page T_i, and d is a damping factor fixed between 0 and 1. Although PageRank could be thought of as an unsupervised machine learning algorithm, most would not classify it as such. Due to the sheer volume of data collected on page visit frequency, click speeds, user feedback, and much more in the recent years, search engines have begun to move away from solely using variations of the classical PageRank algorithm and towards incorporating machine-learned ranking (MLR) into their engines. In 2005 Microsoft researchers unveiled RankNet, a MLR algorithm that utilized features independent from the link structure of the Web to surpass previous versions of PageRank. A web pages' features were condensed into four categories: page-level, domain-level, anchor text and inlinks, and popularity. From these four categories, not only did RankNet outperform PageRank on their ranking accuracy metric, but also in computational time and memory usage [35]. Recently Google has incorporated a machine learning algorithm called RankedBrain into their overall search query algorithm Hummingbird. Although not much is known about the inner workings of RankedBrain, it is theorized it may associate search queries with words correlated to similar queries made by other similar users based on a variety of characteristics such as search history and demographics [7].

One of the most lucrative applications to machine learning is online advertising. There are several different variations online advertising can take including: sponsored search advertising, contextual advertising, and display advertising. In each case, when a user visits a web page or utilizes a search engine, an auction mechanism determines which advertisements to display, what order they are shown in, and what prices the advertisers pay if their advertisement is clicked. As such, a website wants to display the best advertisement to that user in order to maximize the number of advertisement clicks and therefore the money it receives from the advertisers. The ratio of advertisement clicks to displays of the advertisement is referred to as the average click-through rate (CTR) [29]. Companies like Facebook and Google have developed complex machine learning algorithms to optimize this ratio given the vast amount of user information they collect as input data [21]. Figure 1.5 is an infographic reflecting the general method for determining advertisement placement used by Google's advertising service AdWords. When an advertisement is to be generated for a user, first quality scores are created by one of Google's machine learning algorithms for each possible advertisement to be shown. This quality score has components based on CTR, relevance to search query, and landing page. Advertisements with a high enough quality score will be multiplied by the cost-per-click (CPC) bid that the the advertiser is willing to pay in order to show their advertisement. The result is an AD Rank, and the advertisement with the highest AD Rank is then shown to the user. Although the algorithms utilized by these companies are kept private, it is likely that they use machine learning approaches similar to those to solve the multi-armed bandit problem. The multi-armed bandit problem is a problem originally from

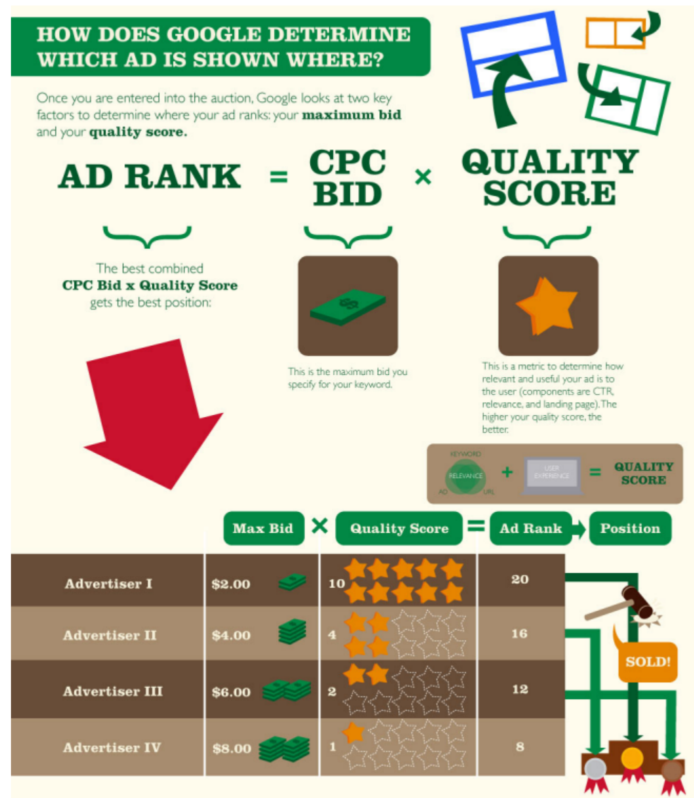


Figure 1.5: Google's Adwords, a method dictating ad placement through the Google search engine. Google holds an auction whose winner is determined by the overall greatest Ad Rank. Ad Rank is the product of an advertisers cost-per-click (CPC) bid and a machine learning algorithm generated quality score. Reproduced from [47].

probability theory in which a gambler is trying to decide which set of k-arms of a series of slot machines (one-armed bandits) to pull in order to maximize their total reward. The gambler must decide what arms to pull, how many pulls of an arm to make, and in what order to pull them all in. As one can see, the multi-armed bandit problem can be seen as a generalized version of this targeted online advertising problem. Thus, methods to optimize the reward from these machines can and have been used to optimize click-through rates. Examples of popular machine learning algorithms to solve this problem include SoftMax and Exp3 [44].

Recommendation or recommender systems is the main machine learning application that will be focused on in this paper. Similar to targeted advertising, recommendation or recommender systems have become increasingly vital to electronic commerce websites and services over the years. It is commonplace for individuals to make choices about something before they personally acquire sufficient information on the subject. Individuals thus rely on advice or recommendations from their peers, reviews, or surveys. Recommendation systems have been created to augment this process for a variety of applications including search queries, books, music, and movies. In simplest terms, a recommender system aims to predict the rating or preference that a user would give to a certain item or service. These systems most commonly use approaches taken from machine learning called collaborative and content filtering. *Collaborative filtering* collects and analyzes large amounts of information on a user's behaviors, activities, or preferences and makes predictions based on similarity to other users. This

data collection can be done explicitly by directly asking a user directly for their feedback or implicitly by observing their views or habits. *Content based filtering* takes descriptors of an item and makes predictions for a user based on the similarity to items they previously rated. Everyday most individuals will contribute to a recommendation system and be recommended something by one. For example, all popular social networking websites such as Facebook and LinkedIn utilize recommendation systems to find user acquaintances. In addition, services such as Google, Amazon, Pandora, Yahoo!, and Netflix are continuously seeking innovative machine learning algorithms to optimize their recommendation systems and increase both their efficiency and revenue [34]. Recommendation systems moved into the public spotlight when the online movie subscription rental service Netflix began a competition titled "“The Netflix Prize” [1]. The competition ran from October 2006 to September 2009, and challenged the machine learning community to develop systems that could surpass the accuracy of their own recommendation system, Cinematch. Netflix encourages its subscribers to rate the movies they watched on a scale of one to five stars, indicating how much they liked or disliked a particular movie. Overall, Cinematch utilized past ratings from individuals, ratings from other users, and both user and item demographic data to predict and recommend movies each subscriber would be interested in. Since Cinematch uses both collaborative and content based filtering, their recommendation system can be considered a hybrid system. Netflix provided groups with over 100 million anonymous movie ratings in the form of disjoint training and testing data sets. Groups would develop systems that learned from the training data and attempt to predict the missing ratings stored in the test data. The group that developed the best system under the constraint that it performed at least 10% better than the prediction accuracy Cinematch could achieve would win a one-million dollar grand prize. The prediction accuracy metric used for this contest was Root Mean Squared Error (RMSE), which is described in Section 2.4. Over the three year period, 41 thousand teams from nearly two-hundred different countries submitted over 44 thousand prediction sets into the contest. Only 650 teams submitted predictions exceeding Cinematch's accuracy, and only two teams exceeded it by 10%. These teams were named *BellKor's Pragmatic Chaos* and *The Ensemble*, both reaching a 10.06% improvement over Cinematch. *BellKor's Pragmatic Chaos* utilized several sophisticated mathematical algorithms to improve their predictions. Those interested can find publicly available descriptions of their algorithms in [24]. Although those specific algorithms are out of the scope of this paper, they encompass fundamental mathematical ideas and theorems that not only underlie most recommendation systems, but also other machine learning prediction algorithms. The two major fundamental ideas that will be discussed are matrix factorization and neighborhood formation, which are described in Section 2.1 and Section 2.3 respectively.

1.4 Key Terms and Notation Review

Definition 1.4.1. Basis: A set of vectors $\{\boldsymbol{v_1}, ..., \boldsymbol{v_n}\}$ in a vector space V is called a *basis* for V if the vectors are linearly independent and the set of vectors spans V.

Definition 1.4.2. Diagonal Matrix: A matrix is said to be *diagonal* if all entries outside the main diagonal are zero. For example:

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

Definition 1.4.3. Eigenvalue and Eigenvector: Let A be a square n-by-n matrix representation of some linear transformation. Then, if there exists a vector $\boldsymbol{x} \in \mathbb{R}^n \neq 0$, such that $A\boldsymbol{x} = \lambda\boldsymbol{x}$. for some scalar λ, then λ is called the *eigenvalue* of A with corresponding *eigenvector* $\boldsymbol{x}$.

Definition 1.4.4. Identity Matrix: A n-by-n *identity matrix* I_n is a diagonal matrix such that $I_n\boldsymbol{x} = \boldsymbol{x}$ for all n-dimensional vectors $\boldsymbol{x}$. For example:

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Definition 1.4.5. L^2-Norm: The L^2-*norm* of a real valued vector $\boldsymbol{x} = [x_1, ..., x_n]$ is given as $\|\boldsymbol{x}\|_2 = \sqrt{x_1^2 + ... + x_n^2}$. The L^2-norm of a real valued matrix A is the sum of the L^2-norms of its columns. If $(\boldsymbol{a_1}, ..., \boldsymbol{a_n})$ are the columns of A, then:

$$\|A\|_2 = \sum_{i=1}^{n} \|\boldsymbol{a_i}\|_2 \ .$$

Definition 1.4.6. Linearly Independent: A set of vectors $\{\boldsymbol{v_1}, ..., \boldsymbol{v_n}\}$ in a vector space V are *linearly independent* if and only if the only linear combination which equals $\mathbf{0}$, is the trivial one. That is, if $a_1\boldsymbol{v_1} + ... + a_n\boldsymbol{v_n} = \mathbf{0}$, then $a_1 = ... = a_n = 0$.

Definition 1.4.7. Orthogonal Matrix: A n-by-n matrix A is said to be *orthogonal* if $AA^T = I$, where A^T is the transpose of A and I is the n-by-n identity matrix. An orthogonal matrix is always invertible, i.e. $A^{-1} = A^T$. For example:

$$AA^T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I_3$$

Definition 1.4.8. Orthonormal Basis: If a set of vectors $\{\boldsymbol{v_1}, ..., \boldsymbol{v_n}\}$ of a vector space V form a basis for V, they form an *orthonormal basis* for V if $\boldsymbol{v_i} \cdot \boldsymbol{v_j} = 0$ when $i \neq j$ and $\boldsymbol{v_i} \cdot \boldsymbol{v_j} = 1$ when $i = j$. In other words, the vectors are mutually perpendicular and each have a length of one.

Definition 1.4.9. Symmetric Matrix: A *symmetric matrix* is a square matrix that satisfies $A^T = A$. For example:

$$A = A^T = \begin{bmatrix} -6 & 1 \\ 1 & 3 \end{bmatrix}$$

Definition 1.4.10. Transpose: The *transpose* of a matrix A is denoted as A^T and has elements given by $A^T_{i,j} = A_{j,i}$. For example:

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

Definition 1.4.11. Vector Space: A *vector space* V is a set of vectors that is closed under finite vector addition and scalar multiplication. An example is the n-dimensional Euclidean space $\mathbb{R}^n$ where each element is represented by a series of n real numbers, scalars are real numbers, addition is done componentwise, and scalar multiplication is multiplication on each term individually.

Chapter 2

Methods for Data Analysis: Recommendation Systems

2.1 Principal Component Analysis

Analyzing large data sets in terms of the relationships between individual points can lead to very slow and memory-intensive calculations. Recommendation systems today must operate on data sets that can be on the order of hundreds of millions of both users and items [21]. This has led individuals to seek methods to extract relevant information from big data sets while both retaining prediction accuracy and speed in terms of computational time. Principal Component Analysis (PCA) is a statistical and mathematical tool that can, with relatively minimal effort, reduce large data sets to those with lower dimensions while retaining the overall simplified structure of the data. The following is a short statistical and mathematical background of PCA.

PCA is popular tool used in unsupervised machine learning and in many other scientific fields to derive a low-dimensional set of features from a large set of variables. When there exists a large set of data containing a number of possibly correlated variables, PCA allows one to summarize the data set with a smaller number of uncorrelated representative variables that together express a large percentage of the variability in the original data set. These representative variables are called *principal components* [21]. Principal components can be thought of as vectors that point in the directions of most variability. Suppose one has a data set stored in a real valued $m \times n$ matrix A, where m is the number of measurement types and n is the number of samples. In terms of a recommendation system, m can be thought of as the total number of items being rated, while n can be thought of as the number of users. Before beginning one may "pre-process" the matrix, or assess the missing values in the data. One could ignore all rows with missing values, likely leading to data distortion, or estimate them. A naive yet time efficient approach is to fill in missing values by some variation of row averages. Other approaches use sophisticated multivariate statistical techniques to do this pre-processing. These advanced approaches are outside the scope of this paper, but examples of them can be found in [27] and [4]. In addition, in order to use PCA one must first center the data, i.e. subtract each column's mean from its respective column. This can be thought of as a translation of the data points that centers them around the origin. If one treats each sample as an m-dimensional vector, then from linear algebra one knows each of the n samples lie in an m-dimensional vector space V which can be spanned by some orthonormal basis. The idea is to find the "most meaningful" basis to re-express the data set. The term most meaningful can depend on the individual data set being analyzed, however, to utilize PCA one must make the assumption that the most meaningful dynamics of the data set are those with the

largest variance. The term *variance* comes from statistics and it measures how far a set of numbers is spread out from its mean. Consider the set $X = \{x_1, x_2, ..., x_k\}$ with zero mean. Then the variance of X is defined as

$$var_X = \sigma_X^2 = \frac{1}{k}\sum_{i=1}^{k} x_i^2 \ . \tag{2.1}$$

With this in mind, one can then define the *covariance* between two equal sized sets. The covariance measures the degree of the linear relationship between two variables. If one considers the two equal sized sets of measurements with zero means $X = \{x_1, x_2, ..., x_k\}$ and $Y = \{y_1, y_2, ..., y_k\}$, then their covariance is defined as

$$cov_{XY} = \sigma_{XY}^2 = \frac{1}{k}\sum_{i=1}^{k} x_i y_i \ . \tag{2.2}$$

Thus, a large positive covariance indicates positively correlated data, a large negative covariance indicates negatively correlated data, and a covariance of zero indicates no correlation in the data. If one stores the data from sets X and Y inside of row vectors $\boldsymbol{x} = [x_1, x_2, ..., x_k]$ and $\boldsymbol{y} = [y_1, y_2, ..., y_k]$ respectively, then one can express the covariance as the dot product

$$cov_{\boldsymbol{xy}} = \sigma_{\boldsymbol{xy}}^2 = \frac{1}{k}(\boldsymbol{x} \cdot \boldsymbol{y}) = \frac{1}{k}\boldsymbol{x}\boldsymbol{y}^T \ . \tag{2.3}$$

One can expand this definition of covariance to the centered measurements stored in the $m \times n$ data matrix A defined previously. Each of the m rows of A can be considered the n measurements of a particular type, or all the ratings for a particular item. With this expansion one arrives at a central concept of PCA, the covariance matrix. The *covariance matrix* is defined as

$$C_A = \frac{1}{n}AA^T \ . \tag{2.4}$$

C_A encompasses the covariance between all pairs of measurements since $C_A = \frac{1}{n}AA^T = (\frac{1}{n}AA^T)^T = C_A^T$, C_A is a square and symmetric $m \times m$ matrix. The ij^{th} element of C_A is the dot product between the vector of the ith measurement type with the vector of the jth measurement type. When $i = j$, $C_{A,ij}$ is the variance of a measurement type. When $i \neq j$, then $C_{A,ij}$ is the covariance between two different measurement types. If a diagonal value is relatively large then there exists an important structure or pattern in the data based on our assumption that large variance is meaningful. If an off-diagonal value is relatively large in absolute value, then there exists a large level of redundancy in the data. This would indicate that one or more variables are directly correlated to another variable. In terms of recommendation systems, knowing about how one item is rated may lead to direct insight on how individuals rate another similar item.

The objective now is to optimize the covariance matrix C_A to both minimize the redundancy in the data as well as retain the underlying structure or signal of the data indicated by the variance. An optimized covariance matrix C_B would have off-diagonal elements of zero, i.e. no redundancy. In other words C_B would be a diagonal matrix and the matrix B would then be decorrelated. In addition, the diagonals of C_B should be ranked in descending order according to variance to see where the most meaningful structures lie in the data. Specifically one must find an orthonormal matrix P such that $B = PA$ under the constraint that $C_B = \frac{1}{n}BB^T$ is a diagonal matrix. The rows of P are then the

principal components of A. Using elementary matrix manipulation techniques, by re-writing C_B we find

$$\begin{aligned} C_B &= \frac{1}{n}BB^T \\ &= \frac{1}{n}(PA)(PA)^T \\ &= \frac{1}{n}PAA^TP^T \\ &= P(\frac{1}{n}AA^T)P^T \\ C_B &= PC_AP^T \ . \end{aligned} \tag{2.5}$$

Thus, it is shown one can relate the optimized covariance matrix C_B to the original covariance matrix C_A and the principal components of A. One may remember that C_A is symmetric, and therefore one can utilize its *symmetric eigenvalue decomposition*.

Theorem: Let A be a real symmetric $n \times n$ matrix with n linearly independent eigenvectors, $\boldsymbol{q_1}, \boldsymbol{q_2}, ..., \boldsymbol{q_n}$. Then A has a *symmetric eigenvalue decomposition* of the form

$$A = Q\Lambda Q^T = Q\Lambda Q^{-1} \tag{2.6}$$

where Q is an orthogonal $n \times n$ matrix whose i^{th} column is the eigenvector $\boldsymbol{q_i}$ of A and Λ is the diagonal matrix whose diagonal elements are the corresponding real eigenvalues, $\Lambda_{ii} = \lambda_i$.

Hence by writing $C_A = Q\Lambda Q^T$, one can write C_B as $P(Q\Lambda Q^T)P^T$. Since Λ is a diagonal matrix, it makes sense to select P to be a matrix where each row $\boldsymbol{p_i}$ is an eigenvector of C_A. Thus, set $P = Q^T$. It then follows that

$$\begin{aligned} C_B &= PC_AP \\ &= P(Q\Lambda Q^T)P^T \\ &= P(P^T\Lambda P)P^T \\ &= (PP^T)\Lambda(PP^T) \\ &= (PP^{-1})\Lambda(PP^{-1}) \\ C_B &= \Lambda \end{aligned} \tag{2.7}$$

One can now conclude that the principle components of A are nothing more than the eigenvectors of $C_A = \frac{1}{n}AA^T$, and that the ith diagonal value of C_B is the variance of A along the ith principal component $\boldsymbol{p_i}$. Even more, we know these eigenvectors are orthogonal to each other since $P = Q^T$ is orthogonal and one can order them by variance. One now has the required information about the data set to reduce its dimensionality while retaining the overall underlying structure. This dimension reduction, however, will be discussed in Section 2.2. Principal Component Analysis and Singular Value Decomposition (SVD) are two intimately related approaches to achieve dimension reduction, and are often called and used interchangeably. SVD is a type of factorization that can be done on any matrix. It is a way to manipulate a data matrix to reveal simpler and more meaningful structures in the data. Techniques in using SVD can lead to a more stable solution and analysis of C_A without actually computing it.

2.2 Singular Value Decomposition

Recommendation systems suffer from a number of fundamental problems which reduces overall quality of predictive ratings such as sparsity and scalability. Storing and comparing users under very large data sets can lead to slow and memory-intensive calculations. This has led individuals to seek methods to decrease the number of dimensions of their data while retaining prediction accuracy. Singular Value Decomposition (SVD), alike Principal Component Analysis (PCA), serves to retain prediction accuracy and speed up computational time when dealing with large data sets stored in matrices. As will be shown later in this section, SVD and PCA are interconnected and the dimensionality reduction techniques done by computing the SVD directly relates to those techniques in PCA.

When attempting to conduct a dimension reduction, one may find it useful to convert the data stored in a matrix into a form that reveals more about the data's structure. First, one must preprocess the data as mentioned in Section 2.1. One may think back to an introductory linear algebra class and attempt to find an eigendecomposition of the matrix. Through *eigendecomposition* one can represent a matrix solely in terms of its eigenvalues and eigenvectors [15].

Theorem: Let A be a square $n \times n$ matrix with n linearly independent eigenvectors, $\boldsymbol{q_1}, \boldsymbol{q_2}, ..., \boldsymbol{q_n}$. Then A has an *eigendecomposition* of the form

$$A = Q\Lambda Q^{-1} \tag{2.8}$$

where Q is a $n \times n$ matrix whose i^{th} column is the eigenvector $\boldsymbol{q_i}$ of A and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, $\Lambda_{ii} = \lambda_i$.

Although an intuitive thought, one can notice that eigendecomposition only works with square matrices. In terms of recommendation systems, rating matrices are almost never square as there are usually more items than users or vice-versa. To work around this, an alternative form of matrix decomposition is used, namely *Singular Value Decomposition* [45].

Theorem: Let A be an arbitrary real $m \times n$ matrix. Then A has a *Singular Value Decomposition* of the form

$$A = USV^T = \sum_{i=1}^{r} s_i \boldsymbol{u_i} \boldsymbol{v_i^T} \tag{2.9}$$

where U and V are two orthogonal matrices with dimensions $m \times m$ and $n \times n$ respectively, and S has diagonal entries consisting of non-negative real numbers in descending order of magnitude. When S has $r \leq n$ zero diagonal entries, it is common to remove the last $n - r$ columns and rows of S to achieve an *economy SVD*. This economy SVD reduces computational time without sacrificing underlying structure. The dimensions of U, S, and V become $m \times r$, $r \times r$, and $r \times n$ respectively in this case.

S is referred to as the *singular matrix* and its diagonal entries, $(s_1, s_2, ..., s_r)$, are called the *singular values* of A. The first r singular values, $(s_1, s_2, ..., s_r)$, are such that $s_i > 0$ and $s_1 \geq s_2 \geq ... \geq s_r$. These first r singular values correspond to the square roots of the non-zero eigenvalues of both AA^T and A^TA. The first r columns of U correspond to the eigenvectors of AA^T and are called the *left singular vectors* of A. The first r columns of V correspond to the eigenvectors of A^TA and are called the *right singular vectors* of A. For each singular value s_i of A there exists a left-singular and right-singular vector $\boldsymbol{u_i}$ and $\boldsymbol{v_i}$ such that $A\boldsymbol{v_i} = s_i\boldsymbol{u_i}$ and $A^T\boldsymbol{u_i} = s_i\boldsymbol{v_i}$ [45].

A reader may notice that SVD provides both the square roots of the eigenvalues as well as the eigenvectors of AA^T. We set the covariance matrix in section 2.1 to be $C_A = \frac{1}{n}AA^T$. Thus, by decomposing A into its SVD, one finds that the principal components of A are equivalent to the left-singular vectors of the SVD of A. Also, the singular values, as defined above, provide a ranking in descending order of the variation in the data alike PCA. One can see this connection mathematically through basic matrix manipulation.

$$\begin{aligned} A &= USV^T, \quad \text{thus} \\ AA^T &= (USV^T)(USV^T)^T \\ &= (USV^T)(VS^TU^T) \\ &= USV^{-1}VS^TU^T \\ &= USS^TU^T \\ AA^T &= US^2U^T, \quad \text{therefore} \\ C_A &= \frac{1}{n}(US^2U^T) \end{aligned} \tag{2.10}$$

A similar manipulation of the matrix A^TA will result in $A^TA = VS^2V^T$. It then follows that if one would like to conduct PCA on A^T, they can simply observe the right-singular vectors of the SVD of A.

SVD and PCA have primarily been used to take a high dimensional and highly variable sets of data points and reduce it to a lower dimensional space. Since, in both cases, the data is ordered based on variation, one can ignore all variation below a defined threshold and still be assured that major relationships in the data will be preserved. In terms of the SVD, [15] proved that no other matrix of rank at most k is closer to A in the L^2-norm than the *rank-k approximation* of A, $A_k = U_kS_kV_k^T$.

Eckart-Young Theorem: Let the SVD of a $m \times n$ matrix A, with rank r, be given by Equation 2.9. If $0 \leq k < r$ and $A_k = U_kS_kV_k^T = \sum_{i=1}^{k} s_iu_iv_i^T$, then

$$\min_{rank(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = s_{k+1} \tag{2.11}$$

A proof by contradiction is the quickest way to prove this theorem. In short, one can make the assumption that there exists a matrix B with $rank(B) \leq k$ that better approximates A. One can show through dimensional analysis of B that for this to be true $(dim(null(B)) + rank(B)) > n$, which is impossible. A full proof of the theorem can be found in [18].

Methods to calculate the SVD of a matrix greatly vary depending on the size and complexity of the matrix. To form the SVD of a $m \times n$ matrix A, one needs the eigenvectors of A^TA and AA^T and the eigenvalues of A^TA or AA^T. This is analogous to how one would conduct PCA on A. When computing the SVD of a relatively small matrix by hand one can follow the following straight forward algorithm [15]:

1. Form A^TA

2. Find the eigenvalues and orthonormalized eigenvectors of A^TA, i.e.,

$$A^TA = V\Lambda V^T$$

3. Sort the eigenvalues in descending order according to their magnitude, and set

$$s_i = \sqrt{\lambda_i}, \text{ for } i = 1 : n$$

4. Find the first r columns of U via

$$\boldsymbol{u_i} = s_i^{-1} A \boldsymbol{v_i}, \text{ for } i = 1 : r$$

5. Find remaining $m - r$ columns such that U is unitary, i.e.,

$$U^T U = U U^T = I$$

Here is an example of this algorithm. Given the matrix

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

1.

$$A^T A = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 9 & 8 \\ 8 & 9 \end{bmatrix}.$$

2. The eigenvectors of $A^T A$ are defined by $A^T A \boldsymbol{v} = \lambda \boldsymbol{v}$, also written as $(A^T A - \lambda I)\boldsymbol{v} = \boldsymbol{0}$. Thus we solve the characteristic equation

$$det(A^T A - \lambda I) = \begin{vmatrix} (9-\lambda) & 8 \\ 8 & (9-\lambda) \end{vmatrix} = 0$$

which becomes

$$\begin{gathered} (9-\lambda)(9-\lambda) - 8 \cdot 8 = 0 \\ \lambda^2 - 18\lambda + 17 = 0 \\ (\lambda - 17)(\lambda - 1) = 0. \end{gathered}$$

Thus the eigenvalues of $A^T A$ are $\lambda_1 = 17$ and $\lambda_2 = 1$. Plugging λ_1 back into $(A^T A - \lambda I)\boldsymbol{v} = \boldsymbol{0}$ and solving for $\boldsymbol{v_1}$ we get the system of equations

$$\begin{gathered} (9-17)\boldsymbol{v_1}(1) + 8\boldsymbol{v_1}(2) = -8\boldsymbol{v_1}(1) + 8\boldsymbol{v_1}(2) = 0 \\ 8\boldsymbol{v_1}(1) + (9-17)\boldsymbol{v_1}(2) = 8\boldsymbol{v_1}(1) - 8\boldsymbol{v_1}(2) = 0 \end{gathered}$$

and we find,

$$\boldsymbol{v_1}(1) = \boldsymbol{v_1}(2) \quad .$$

Thus we choose the normalized vector

$$\boldsymbol{v_1} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}.$$

By plugging in λ_2 and following the same procedure, we find

$$\boldsymbol{v_2} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}.$$

Now the matrix V can be constructed

$$V = \begin{bmatrix} \boldsymbol{v_1} & \boldsymbol{v_2} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}.$$

3. $s_1 = \sqrt{\lambda_1} = \sqrt{17}$ and $s_2 = \sqrt{\lambda_2} = 1$, so that

$$S = \begin{bmatrix} \sqrt{17} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

4. The first two columns of U can be computed with

$$\boldsymbol{u_1} = \tfrac{1}{\sqrt{17}} A\boldsymbol{v_1} = \tfrac{1}{\sqrt{17}} \begin{bmatrix} 1 & 2 \\ 2 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \tfrac{1}{\sqrt{34}} \begin{bmatrix} 3 \\ 4 \\ 3 \end{bmatrix}$$

and

$$\boldsymbol{u_2} = \tfrac{1}{1} A\boldsymbol{v_2} = \tfrac{1}{1} \begin{bmatrix} 1 & 2 \\ 2 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} = \tfrac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}.$$

Thus a partial construction for U is

$$U = \begin{bmatrix} \boldsymbol{u_1} & \boldsymbol{u_2} & \boldsymbol{u_3} \end{bmatrix} = \begin{bmatrix} \frac{3}{\sqrt{34}} & \frac{-1}{\sqrt{2}} & \boldsymbol{u_3}(1) \\ \frac{4}{\sqrt{34}} & 0 & \boldsymbol{u_3}(2) \\ \frac{5}{\sqrt{34}} & \frac{1}{\sqrt{2}} & \boldsymbol{u_3}(3) \end{bmatrix}.$$

5. U must be unitary, which implies $\boldsymbol{u_3}$ must satisfy

$$\boldsymbol{u_i} \cdot \boldsymbol{u_3} = I_{i3}, \text{ for } i = 1, 2, 3$$

and thus,

$$\boldsymbol{u_3} = \tfrac{1}{\sqrt{17}} \begin{bmatrix} 2 \\ -3 \\ 2 \end{bmatrix}.$$

In which we arrive at

$$A = USV^T = \begin{bmatrix} \frac{3}{\sqrt{34}} & \frac{-1}{\sqrt{2}} & \frac{2}{\sqrt{17}} \\ \frac{4}{\sqrt{34}} & 0 & -\frac{3}{\sqrt{17}} \\ \frac{5}{\sqrt{34}} & \frac{1}{\sqrt{2}} & \frac{2}{\sqrt{17}} \end{bmatrix} \begin{bmatrix} \sqrt{17} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}.$$

The above algorithm should work in theory for all real matrices, however in practice it is solely used for small matrices that can be done by hand. When a computer calculates the covariance matrix (either through A^TA or AA^T) of a matrix A on the order of millions or trillions of data points, numerical rounding error in the eigenvalues and eigenvectors is almost always guaranteed. For this reason individuals have created methods to compute the SVD of a matrix without directly calculating the covariance matrix. This is the largest factor for many scientists to utilize SVD within or as an alternative to PCA. There are a great deal of algorithms to compute the SVD each with their own optimizations for different types of data sets. Some utilize numerical iterative methods such as classical bisection method or the Guass-Seidel method [8] within their algorithms. MATLABs basic function *svd* computes the singular value decomposition of a real valued matrix by calling a routine called SGESVD from the LAPACK library. The routine has some modifications in case singular values are extremely small, but implements variations of the following algorithms in tandem: Householder reduction and Golum-Reinsch SVD [8]. The documentation for SGESVD can be found in [2], but pseudo-code for both the Householder reduction and Golum-Reinsch SVD can be found in Figure 2.1.

2.3 Neighborhood Formation

At the core of every recommendation system is the idea of similarity within users and or items. For example, if a certain movie is highly rated by most individuals, then the system may recommend new movies with similar characteristics, such as genre or starring actors, to that item. The same goes for users. If a series of users with the same demographics really enjoy a certain type of movie, then the system may recommend that movie to other users with similar demographics. In each case, there needs to be a way to find if there is a dependence in the characteristic data collected that can be used to relate certain items or users together. In other words, one needs to find the nearest neighbors or form a neighborhood around each user or item that are most similar to them in order to maximize overall predictive accuracy. A group of users or items with the greatest similarity to one another form a neighborhood. The methods for computing the similarity between two items or two users are intrinsically the same. Given a ratings matrix A, one can calculate the similarity, sim_{ij} between two arbitrary items, i and j, by first isolating users that have rated both of these items. Then one applies a similarity computation technique over all of these similar users and arrives at sim_{ij}. This process is illustrated in Figure 2.2. Any similarity computation technique can also utilize the rank-k approximation of A as described in Section 2.2. By choosing a number of singular values k that captures some majority of the variance in the ratings data, one can significantly reduce the amount of pair comparisons made when calculating similarities. By forming $A_k = U_k S_k V_k^T$, one can then form $Q_{items} = \sqrt{S_k} V_k$, the k-by-n matrix that represents all n items in a k dimensional space of psuedo-users. Since k tends to be significantly smaller than the total number of users that rated each pair of items, the overall similarity computational time is drastically reduced while prediction accuracy is maintained. It should be said that a similar matrix, $P_{users} = U_k \sqrt{S_k}$, representing n psuedo-items in a k dimensional space can also be used for finding similarity in users. Whether a system chooses to utilize rank-k approximation or not, recommender systems typically use one of three different similarity computation techniques. These are Pearson correlation, cosine-based similarity, and adjusted cosine similarity [37].

The Pearson correlation is a classical method for looking for dependence between variables. From statistics, the *Pearson correlation* measures the linear relationship between two variables. The measure produced, the Pearson correlation coefficient, is between positive and negative one inclusive. One indicates a total positive correlation, zero indicates no correlation, and negative one is total negative

Algorithm 1a: Householder reduction to bidiagonal form:
Input: m, n, A where A is $m \times n$.
Output: B, U, V so that B is upper bidiagonal, U and V are products of Householder matrices, and $A = UBV^T$.

1. $B \leftarrow A$. (This step can be omitted if A is to be overwritten with B.)
2. $U = I_{m \times n}$.
3. $V = I_{n \times n}$.
4. For $k = 1, \ldots, n$
   a. Determine Householder matrix Q_k with the property that:
      - Left multiplication by Q_k leaves components $1, \ldots, k-1$ unaltered, and
      - $Q_k \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b_{k-1,k} \\ b_{k,k} \\ b_{k+1,k} \\ \vdots \\ b_{m,k} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b_{k-1,k} \\ s \\ 0 \\ \vdots \\ 0 \end{bmatrix}$, where $s = \pm\sqrt{\sum_{i=k}^{m} b_{i,k}^2}$.

   b. $B \leftarrow Q_k B$.
   c. $U \leftarrow U Q_k$.
   d. If $k \leq n-2$, determine Householder matrix P_{k+1} with the property that:
      - Right multiplication by P_{k+1} leaves components $1, \ldots, k$ unaltered, and
      - $\begin{bmatrix} 0 & \cdots & 0 & b_{k,k} & b_{k,k+1} & b_{k,k+2} & \cdots b_{k,n} \end{bmatrix} P_{k+1} = \begin{bmatrix} 0 & \cdots & 0 & b_{k,k} & s & 0 & \cdots & 0 \end{bmatrix}$, where $s = \pm\sqrt{\sum_{j=k+1}^{n} b_{k,j}^2}$.

   e. $B \leftarrow B P_{k+1}$.
   f. $V \leftarrow P_{k+1} V$.

Algorithm 1b: Golub–Reinsch SVD:
Input: m, n, A where A is $m \times n$.
Output: Σ, U, V so that Σ is diagonal, U and V have orthonormal columns, U is $m \times n$, V is $n \times n$, and $A = U \Sigma V^T$.

1. Apply Algorithm 1a to obtain B, U, V so that B is upper bidiagonal, U and V are products of Householder matrices, and $A = UBV^T$.
2. Repeat:
   a. If for any $i = 1, \ldots, n-1$, $|b_{i,i+1}| \leq \varepsilon \left(|b_{i,i}| + |b_{i+1,i+1}|\right)$, set $b_{i,i+1} = 0$.
   b. Determine the smallest p and the largest q so that B can be blocked as

$$B = \begin{bmatrix} B_{1,1} & 0 & 0 \\ 0 & B_{2,2} & 0 \\ 0 & 0 & B_{3,3} \end{bmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$

   where $B_{3,3}$ is diagonal and $B_{2,2}$ has no zero superdiagonal entry.
   c. If $q = n$, set $\Sigma =$ the diagonal portion of B **STOP**.
   d. If for $i = p+1, \ldots, n-q-1$, $b_{i,i} = 0$, then
         Apply Givens rotations so that $b_{i,i+1} = 0$ and $B_{2,2}$ is still
         upper bidiagonal. (For details, see [GL96, p. 454].)
      else
         Apply Algorithm 1c to n, B, U, V, p, q.

Algorithm 1c: Golub–Kahan SVD step:
Input: n, B, Q, P, p, q where B is $n \times n$ and upper bidiagonal, Q and P have orthogonal columns, and $A = QBP^T$.
Output: B, Q, P so that B is upper bidiagonal, $A = QBP^T$, Q and P have orthogonal columns, and the output B has smaller off-diagonal elements than the input B. In storage, B, Q, and P are overwritten.

1. Let $B_{2,2}$ be the diagonal block of B with row and column indices $p+1, \ldots, n-q$.
2. Set $C =$ lower, right 2×2 submatrix of $B_{2,2}^T B_{2,2}$.
3. Obtain eigenvalues λ_1, λ_2 of C. Set $\mu =$ whichever of λ_1, λ_2 that is closer to $c_{2,2}$.
4. $k = p+1, \alpha = b_{k,k}^2 - \mu, \beta = b_{k,k} b_{k,k+1}$.
5. For $k = p+1, \ldots, n-q-1$
   a. Determine $c = \cos(\theta)$ and $s = \sin(\theta)$ with the property that:

$$\begin{bmatrix} \alpha & \beta \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} \sqrt{\alpha^2 + \beta^2} & 0 \end{bmatrix}.$$

   b. $B \leftarrow B R_{k,k+1}(c, s)$ where $R_{k,k+1}(c, s)$ is the Givens rotation matrix that acts on columns k and $k+1$ during right multiplication.
   c. $P \leftarrow P R_{k,k+1}(c, s)$.
   d. $\alpha = b_{k,k}, \beta = b_{k+1,k}$.
   e. Determine $c = \cos(\theta)$ and $s = \sin(\theta)$ with the property that:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \sqrt{\alpha^2 + \beta^2} \\ 0 \end{bmatrix}.$$

   f. $B \leftarrow R_{k,k+1}(c, -s) B$, where $R_{k,k+1}(c, -s)$ is the Givens rotation matrix that acts on rows k and $k+1$ during left multiplication.
   g. $Q \leftarrow Q R_{k,k+1}(c, s)$.
   h. if $k \leq n-q-1 \alpha = b_{k,k+1}, \beta = b_{k,k+2}$.

Figure 2.1: Pseudo-code using both the Householder Reduction (1a) and Golum-Reinsch (1b & 1c) algorithms for decomposing a matrix into its Singular Value Decomposition (SVD). These algorithms form the most popular basis for other advanced methods for finding the SVD of a matrix quickly and efficiently. Taken from [25].

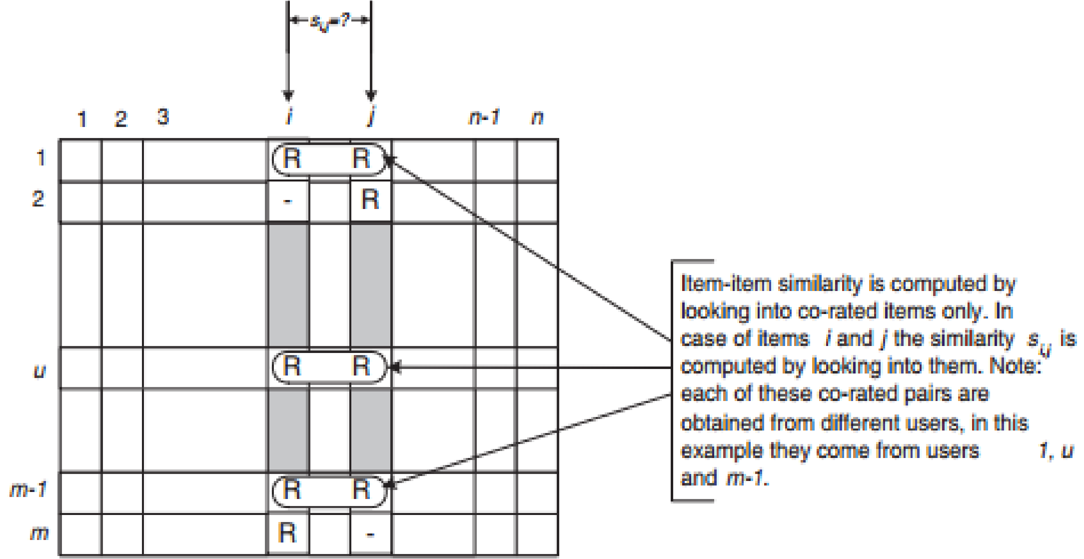


Figure 2.2: The isolation of co-rated items and similarity computation for neighborhood formation. Ratings from m users and n items are stored in an m-by-n matrix. The similarity of items i and j are computed by comparing ratings by users that rated both items. These ratings are then input into a similarity computation such as the Pearson or Cosine correlation. Taken from [37].

correlation. This correlation is calculated by dividing the covariance of the two variables by the product of their standard deviations. The covariance between two variables is calculated by Equation 2.2, and the standard deviation of a variable is given by the square root of the variance given by Equation 2.1. In terms of a ratings matrix of data, the Pearson similarity of two items can be described by

$$sim_{i,j} = \rho_{i,j} = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}} \tag{2.12}$$

where R is either the ratings matrix A or the pseudo-user matrix $Q_{items} = \sqrt{S_k}V_k$, U is the set of all co-rated users u of items i and j of R, and $\bar{R}_i$ is the average rating of the ith item in terms of R.

Another form of similarity computation is cosine-based similarity. *Cosine similarity* is a measure of similarity between two vectors that computes the cosine of the angle between two vectors. It too ranges between positive and negative one alike the Pearson correlation. If one thinks of the two items, i and j, as being represented by two vectors, $\boldsymbol{i}$ and $\boldsymbol{j}$, in an m or k dimensional user-space stored in A or Q_{items} respectively, one can compute the cosine similarity between any two items by

$$sim_{ij} = cos_{\boldsymbol{ij}} = \frac{\boldsymbol{i} \cdot \boldsymbol{j}}{\|\boldsymbol{i}\|_2 \|\boldsymbol{j}\|_2} \quad . \tag{2.13}$$

One does not need to first isolate the co-rated cases for cosine similarity as performing the dot product on vectors $\boldsymbol{i}$ and $\boldsymbol{j}$ results in an addition of zero for any non co-rated case. Although cosine similarity can be used for user-based neighborhood formation, it is not an optimal method for forming item-based neighborhoods. Individual users tend to have different rating biases when rating a collection of items. For example, on average a user may rate items significantly higher than another user although the two users may have the same tastes. Thus, *adjusted cosine similarity* is used to offset this drawback

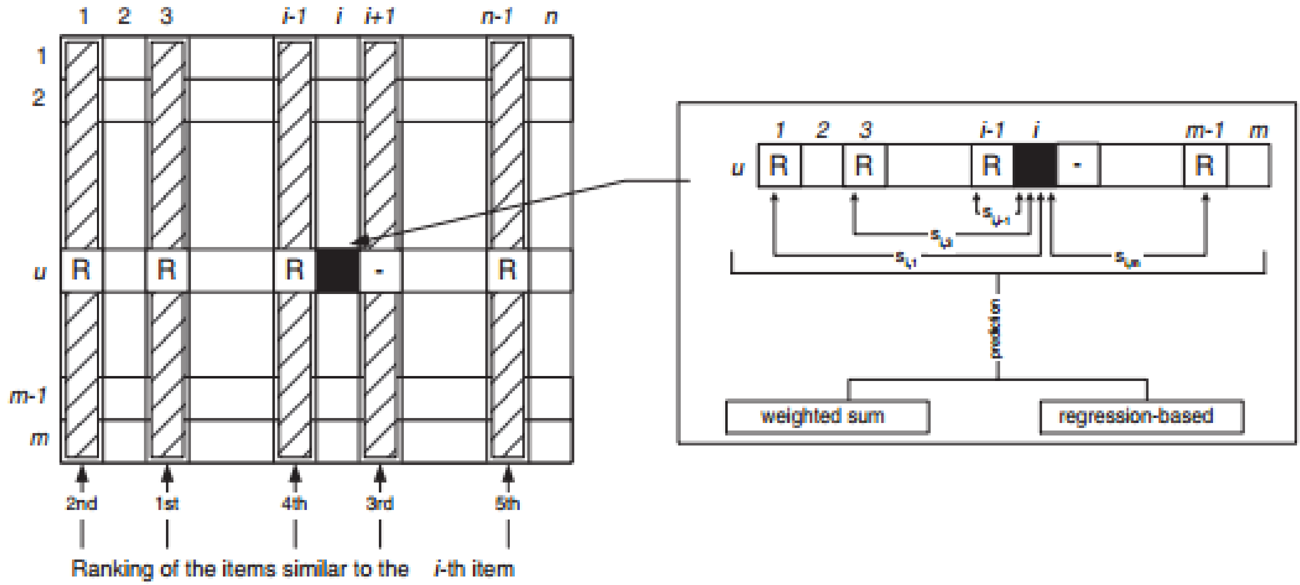


Figure 2.3: The process for predicting a rating for an item i for a user u using item-based collaborative filtering. Known ratings are stored in a matrix with m users and n items. The similarity $sim_{i,j}$ is used as a weight for each neighbor's rating where j is one of the five neighbors pictured. The sum of these weighted ratings are then averaged and form the prediction for item i for user u. Reproduced from [37].

by subtracting the corresponding user average from each co-rated item pair. It then follows that one must again isolate all co-rated cases as in the Pearson correlation. Utilizing the same descriptors as for Equation 2.12, the formal equation for the adjusted cosine similarity is given by

$$sim_{i,j} = adjcos_{i,j} = \frac{\sum_{u \in U}(R_{u,i} - \overline{R_u})(R_{u,j} - \overline{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \overline{R_u})^2}} \tag{2.14}$$

where $\overline{R_u}$ is the average of the uth user's ratings.

With one of these methods to compute the similarity between items in hand, one can formulate predictions on a non-rated item i for some user u. This can be done by restricting the number of neighbors an item should have and using the similarity between the item and those neighbors as weights. Each rating is weighted by the corresponding similarity sim_{ij} for items i and j. This process is pictured in Figure 2.3. A basic prediction for $pred_{ui}$ for an item i of a user u can be given by

$$pred_{u,i} = \frac{\sum_{n \in N_i}(sim_{i,j} * R_{u,n})}{\sum_{n \in N_i}|sim_{i,j}|} \tag{2.15}$$

where N_i is the set of all neighbors of item i.

2.4 Predictive Accuracy Metrics

Prediction accuracy is a frequently discussed topic within recommendation system literature. The ultimate goal of any recommender is, in some form, to produce accurate predictions of how a user

will view a given item. One of the fundamental assumptions of a recommendation system is that the more accurate the predictions it can produce about a user, the more likely that user will prefer that system and the services it may provide. As such, creators of these systems search for algorithms that can provide the best predictions. However, the methods to evaluate various algorithms are inherently difficult. For example, different algorithms may perform very well on one specific data set, but poorly on another. In addition, the specific task or goals of a prediction system can differ. It may be very valuable that a system minimizes leading a user to a wrong choice, while another may want to provide a descriptive yet simplified explanation of a recommendation to a user. These goals may be mutually exclusive and one metric may fail to capture a system's unique goals. As such, I will be limiting this paper to two classes of popularly used accuracy measures as identified in [22] and [39]. These two general classes are: ratings prediction measures, and usage or ranking prediction measures.

Ratings prediction is when one purely wants to predict the ratings a user would give to an item. This is the most common used version of predictive accuracy metrics. The most widely used ratings prediction measures involve some variation of the *Mean Squared Error (MSE)*. A recommender system will generate predicted ratings r'_{ui} for some test set of user-item pairs (u, i) for which the actual ratings r_{ui} are known yet hidden from the system while predicting. This test set can be represented by a matrix $T \subseteq A$ where A is a m-by-n (user-by-item) known ratings matrix and $A_{ui} = r_{ui}$. With this information, the MSE is calculated by

$$MSE = \frac{1}{|T|} \sum\nolimits_{(u,i) \in T} (r_{ui} - r'_{ui})^2 \tag{2.16}$$

where $|T|$ is the number of rated elements in the test set T. The most common variation of MSE is the *Root Mean Square Error (RMSE)*. RMSE is simply the square root of MSE and is given by

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{|T|} \sum\nolimits_{(u,i) \in T} (r_{ui} - r'_{ui})^2} \ . \tag{2.17}$$

A popular alternative to MSE and RMSE is the *Mean Absolute Error (MAE)*. The MAE is calculated by

$$MAE = \frac{1}{|T|} \sum\nolimits_{(u,i) \in T} |r_{ui} - r'_{ui}| \ . \tag{2.18}$$

One can also produced normalized versions of these measures known as *Normalized RMSE (NM-RMSE)* and *Normalized MAE (NMAE)*. These alternative measures are produced by dividing their respective measures by the range of the data set being used $(r_{max} - r_{min})$. NMRMSE and NMAE are used commonly to allow comparisons between data sets with different rating scales to be made more clearly. The MSE and RMSE measures penalize large errors by squaring each individual error. Thus these measures will be minimized on systems which have small inaccuracies over several predictions rather than only a few large inaccuracies over a few. On the other hand, MAE penalizes consistently small inaccuracies over a large set of predictions. To see this difference a small example is provided. Suppose a *user*-by-*item* known ratings matrix A and a predicted ratings matrix A' on a 1-5 rating scale are given by

$$A = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 1 & 3 \\ 4 & 5 & 2 \end{bmatrix}, \ A' = \begin{bmatrix} 2 & 3 & 2 \\ 2 & 2 & 2 \\ 4 & 4 & 1 \end{bmatrix}.$$

We find that these matrices produce an RMSE and NMRMSE of 0.8819 and 0.2205 respectively, and an MAE and NMMAE of 0.7778 and 0.1945 respectively. Since the the MAE measure is less than the RMSE, we are lead to believe that the recommender system that generated these predicted values under this data set produces ratings consistently close to the actual ratings rather than producing relatively large errors over a small subset of ratings.

Usage prediction attempts to recommend items to users that they may be interested in. The predictive ratings of these movies are not what is important, but rather if the user will use these selected items. For example, if a user watches a particular movie, a movie recommender may provide a list of similar movies the user may like to watch. To evaluate usage prediction it is common to select a random test user, hide a subset of their ratings, and use a recommender to predict a set of items the user will actually use. Thus, there will be four possible scenarios for the recommendations and the hidden items as shown in Table 2.1.

In order to conduct usage recommendation evaluation with the above classification one must assume that unused items would not be used even if they had been recommended to the user. In practice this may be false and a user may in fact use an item that they were incorrectly recommended. Thus, the total number of false positives (fp) can be over estimated. In addition, in order to evaluate the performance of recommender systems that generate predictive ratings, one may choose to map these ratings into a binary scale to predict the usage of an item by a user. For example, if a system predicts ratings on a 1-5 scale, a map may be such that items rated 1-3 by a user would be considered not to be used, and 4-5 would be used.

	Recommended	Not Recommended
Used	True Positive (tp)	False Negative (fn)
Not Used	False Positive (fp)	True Negative (tn)

Table 2.1: A classical recommender item usage table. For every potential item that can be recommended to some user, it can be classified by this table depending on whether the recommendation system did or did not recommend the item as well as if it was actually used by a user or not.

Given the items a known user has used in a group of items, one can evaluate the usage recommendation of a system through first counting the number of each tp, fn, fp, and tn that occurred in the testing phase. These will be denoted as TP, FN, FP, and TN. With this information one can then compute the following three quantities.

$$\mathbf{Precision} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}} \tag{2.19}$$

$$\mathbf{Recall} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}} \tag{2.20}$$

$$\textbf{False Positive Rate} = \frac{\textbf{FP}}{\textbf{FP} + \textbf{TN}} \tag{2.21}$$

Precision, as shown in Equation 2.19 is the ratio of used items chosen for recommendation to all items recommended. Precision is thus the measure for efficiency of used item selection and is commonly cited as the most useful measure of interest [39]. Recall, shown in Equation 2.20, is the ratio of used items recommended to all used items. The False Positive Rate, shown in Equation 2.21, is simply the ratio of unused recommended items to all unused items.

When the size of the amount of potential item recommendations presented to the user is not fixed, it is useful to evaluate an algorithm over a range of recommendation list lengths. When this is done one can generate *Precision-Recall Curves* that compare precision to recall as well as *Receiver Operating Characteristic (ROC) Curves* which are taken from signal detection theory to compare true positive rates to false positive rates [39]. The selection of which curve to use is once again based on the overall goal of the application. There are several measures to summarize these curves into a single metric. The most common methods are the *F-measure* of the Precision-Recall Curves and the *Area Under the ROC Curve (AUC)*. An in depth construction of these curves and summarizing metrics can be found in [46] and [36].

Chapter 3

Results: MATLAB Models

3.1 Coded Algorithm Descriptions

In order to apply the knowledge gained from writing this paper, we coded two recommendation algorithms in MATLAB. The two algorithms are "item_SVD_recommender" (ItemSvdRec) and "item_SVD_demo_recommender" (ItemSvdDemoRec), and their code can be found in Section A.1 and Section A.2 respectively. Both of these algorithms are modifications of the pseudo-code described in the paper *Applying SVD on Generalized Item-based Filtering* [45]. As their names suggest, both of these algorithms utilize collaborative and content-based item filtering enhanced by SVD to predict the ratings that users will give to certain items.

ItemSvdRec requires a m-by-n ratings matrix (ratings) with m-users and n-items, a number of singular values ($k \ll m$) to create the kth SVD approximation of the ratings matrix, and the neighborhood size (num_neigh) for the number of neighbors each item can have. The ratings matrix can have any real valued element to indicate a rating, but missing or non-rated elements must be of the form 'NaN.' If this ratings matrix is labeled as A then element $A_{i,j}$ refers to the rating, or lack there of, user i gave to item j. The code provided in Section A.1 is well documented, but the following is a short overview of how the algorithm operates. ItemSvdRec works by first pre-processing the rating matrix to eliminate all non-rated data points. It does this by replacing all NaN values with their corresponding column or item average (col_avg). Then individual user bias is removed by subtracting the corresponding row or user average (row_avg) from each row. This creates a normalized ratings matrix (ratings_norm) which is then reduced to a kth SVD approximation in order to reduce dimensionality. The matrix Q_{items}, as described in Section 2.3, is then created representing the meta ratings given by each of the k pseudo-users. Neighborhood formation is done through conducting the Adjusted Cosine Similarity between each pair of items stored in Q_{items}. A n-by-n item similarity matrix (sim_matrix) is produced at the end of this formation. Finally, prediction generation is conducted and results in a completely filled predicted ratings matrix (ratings_predicted). ItemSvdRec has been optimized in MATLAB to utilize a function called bsxfun, which applies an element-by-element binary operation specified by various function handles on two arrays. This enables ItemSvdRec to generate predictions for all users one item at a time rather than generating predictions for each user-item pair one at a time. Although this speeds up the algorithm, it makes the code difficult to interpret at first glance. This prediction generation begins at line 68, however, the generalized prediction for user u_i on item i_j can be described by the following weighted sum:

$$ratings_predicted(i,j) = \frac{\sum_{n \in N_j} sim_matrix(i,n) * (ratings_norm(i,n) + row_avg(i))}{\sum_{n \in N_j} |sim_matrix(i,n)|} \quad (3.1)$$

where N_j is the set of all neighbor indices of item i_j.

ItemSvdDemoRec is very similar to ItemSvdRec, but it includes the additional feature of utilizing item demographics to increase overall ratings prediction. ItemSvdDemoRec takes as parameters a ratings matrix (ratings), a number of singular values (k) to conduct SVD on ratings, and an item neighborhood size (num_neigh) as described above. In addition, it takes in a x-by-n demographics matrix (item_demographics) where x is the number of item features or characteristics each of the n items have. Each element of this matrix must be either a 1, indicating an item has a specific feature, or a 0, an item does not possess that specific feature. This matrix must contain no missing values. In addition, ItemSvdDemoRec estimates this demographics matrix utilizing a dth SVD approximation with $d \ll x$ singular values. The overall algorithm's procedure is alike ItemSvdRec, but differs on neighborhood formation. Once the dth SVD approximation is completed, a demographic version of Q_{items} is created representing the d demographic meta features of the n items. Since there is no need to remove bias in the demographic features, demographic correlation is done through a Pearson Correlation and stored in a matrix (demo_cor_matrix). The similarity matrix (sim_matrix) is then calculated by the sum of the similarities in rating history of pairs of items and the demographic similarities of the same pairs of items stored in demo_cor_matrix. This results in item neighbors similar to one another in terms of demographics as well as rating histories. The generalized prediction for user u_i on item i_j is also described by Equation 3.1.

3.2 MovieLens Data Sets

In order to utilize our coded machine learning algorithms, we needed to select data sets that were well documented, organized, and accurate. Of the many publicly available data sets, we chose to use a subset of data from the popular MovieLens System. The MovieLens online recommender system is a website where users provide personal movie ratings in order to receive personalized movie recommendations. Since 1997, MovieLens has collected user ratings data and stored them in various data sets depending on the time period collected and the total number of provided ratings. These data sets have been widely used in education, research, as well as in industry. In 2014 the MovieLens data sets were downloaded over 140,000 times alone, and were referenced in over 7,500 pieces of research literature [20]. For our models we chose to look at the MovieLens 100K data set, and the MovieLens 1M data set. These data sets are publicly available to download from the MovieLens website [19]. We believed these data sets to be large enough to find underlying structure in the data, yet small enough to complete our computations in a reasonable amount of time.

The MovieLens 100K data set was released in 1998 and contains rating, item, and user data contained in a downloadable zip-file, "ml-100k.zip." The ratings data is contained in "u.data" and encompasses 100,000 ratings from 943 users and 1682 movies. Movie preference is provided by a rating ranging from integers 1 to 5, 1 meaning a strong dislike and 5 a strong like. Each user in the data set has rated at least 20 movies. This data is provided in the form of tab separated lists of 'user id |item id |rating |timestamp.' Since this data has a small sparsity or coverage of 6.3%, as is depicted in Figure 3.1, a tab separated list of values minimizes storage space over a matrix. All MovieLens ratings data

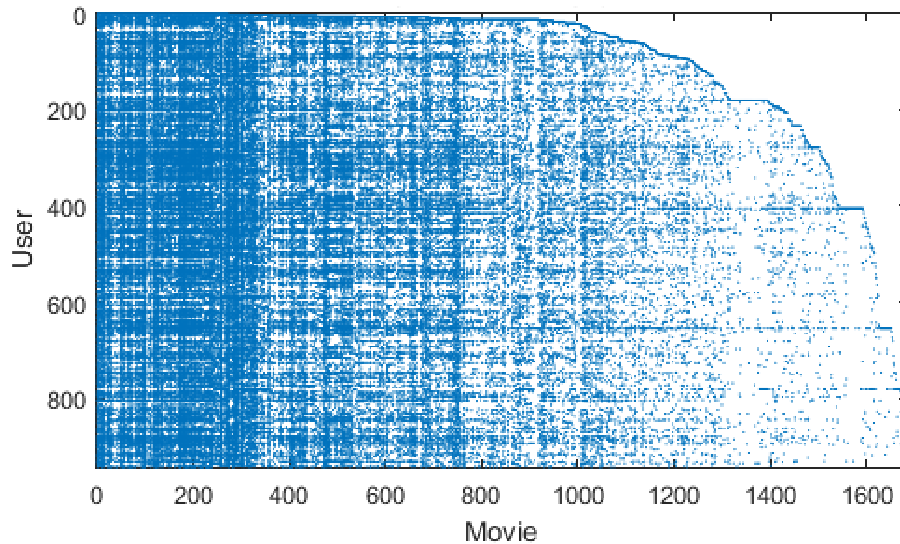


Figure 3.1: Sparsity of the MovieLens 100K data set with 100,000 movie ratings. A blue dot indicates a rated movie by a particular user. The data set has a 6.3 percent sparsity or in other words, 6.3 percent of all possible ratings of movies that can be given by all users is known. This is an average sparsity many movie recommendation systems must operate over.

is stored in a similar fashion to "u.data," and manipulation of the data into matrix form needed to be done prior to running our recommendation algorithms. As an example, the code to download and manipulate "u.data" is provided in Section A.3.

In addition to ratings, the MovieLens 100K data set contains demographic information. Item demographics are stored in "u.item," and are also in a tab separated list. Item demographics consist of movie id, movie title, release date, video release date, IMDb URL, and a list of movie genres. A movie's genre is described by one or more of 19 genre fields such as Action, Comedy, Drama, Mystery, etc. Of these demographics, only movie id, movie title, and movie genres were incorporated to utilize in our ItemSvdDemoRec algorithm. Once again, pre-processing needed to be conducted to store these demographics into a matrix. Each column of this demographics matrix becomes a 19-by-1 demographic feature vector for each movie. A 1 in a demographic feature field indicates the movie is of that genre, while a 0 indicates it is not. Movies can be in several genres at the same time. It should be mentioned that there also exists user demographic information such as age, gender, and occupation, but this was not utilized in either of our algorithms.

Training and testing data subsets are also provided by MovieLens in the 100K zip file. It is common to conduct a five fold cross validation when evaluating the performance of a recommendation algorithm. If one tries to evaluate a recommendation system against the data it learned from, the result could lead to overfitting. *Overfitting* occurs when a statistical model describes the noise in the data instead of the underlying relationship [12]. With five fold cross validation one can avoid overfitting by evaluating against new yet already known data. This validation process requires five random and disjoint testing subsets each representing 20% of all known recommendation data. For each testing subset, the remaining 80% of the data is used as training data for the system. The system is then evaluated by some test metric over the testing subset. The five evaluations are then averaged and the result is considered to be an accurate measure of prediction performance. MovieLens stores sample training and testing data

sets for the 100K ratings in "u1.base" and "u1.test" through "u5.base" and "u5.test."

The MovieLens 1M data set is very similar to the 100K data set. As more data was collected over the years some users changed some of their previous ratings, but the overall known rating data contained in the 100K data set is enveloped in the 1M data set. The data is stored in "ml-1m.zip" and is also freely downloadable from the MovieLens website [19]. Ratings are stored in colon separated format within "ratings.data." As such, pre-proccessing needed to be done to store the data into a matrix. The data set contains 1,000,209 ratings from 6040 users over 3952 movies. Ratings are once again on a 1 to 5 integer scale and each user has rated at least 20 movies. Although there are roughly ten times the number of ratings stored in the MovieLens 1M data set compared to the 100K data set, the sparsity of the data matrix is actually slightly smaller at 4.2%. This is because there is a significant increase in both users and movies between the two data sets.

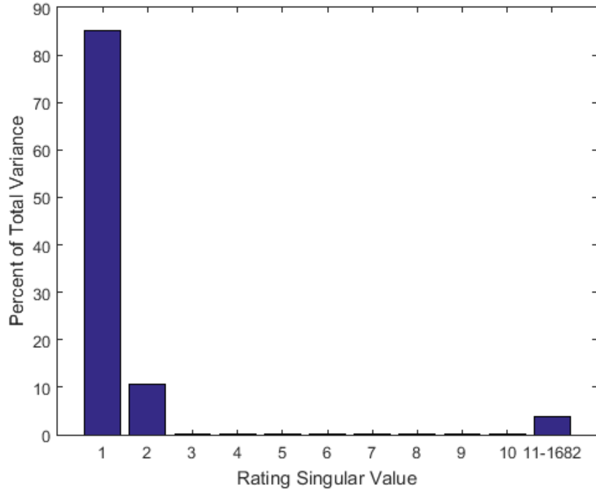
Movie demographic data is again provided in the MovieLens 1M data set. This data can be found in "movies.data" and is in the form of colon separated values. The movie id, title, and list of genres are given. The genres are the same 19 found in the MovieLens 100K data set. The same pre-processing methods were used to change this item demographic data into a usable format.

Unlike the 100K data set, the 1M data set does not contain training and testing subsets to conduct a five fold cross validation. In order to avoid overfitting and compare data sets in the same manner, we developed our own code to generate both testing and training data. In short, a random permutation of all rated entry indices are generated, which are then separated into five disjoint intervals, and finally the known ratings with those indices are then populated into five fully non-rated (NaN) testing matrices. Training data matrices were then created by removing the rated entries in each respective testing matrix from a copy of the original ratings matrix. The code for generating these five fold cross validation training and testing data sets can be found in Section A.4.

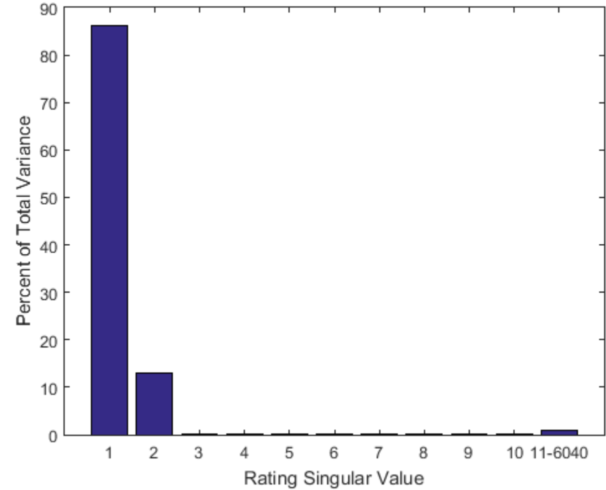
3.3 Prediction Evaluation Metric Results

Before beginning prediction evaluation, we wished to look at the total variance captured by the singular values of both the training rating and demographic data matrices. These singular values play a vital role in the parameter tuning and therefore also the overall performance, under various metrics, of the recommender system. After all missing rating values are filled in during the pre-processing phase of the algorithms, roughly $85.06\% \pm 0.53\%$ and $87.14\% \pm 0.74\%$ of variance in the training rating subsets of MovieLens 100K and 1M respectively were captured in the first singular value. The second singular values captured $11.12\% \pm 0.37\%$ and $12.10\% \pm 0.49\%$ of the variance, indicating about 96% and 99% of the data's variance are stored in the first two singular values. After the second singular value, the variance quickly drops off to less than 0.1% for each of the remaining singular values. The variance captured in the demographic data was much more dispersed among the singular values compared to the rating training data. The first singular values capture 28.26% and 29.52% of the demographic information of the MovieLens 100K and 1M demographic data sets respectively. In comparison, one would need 15 demographic singular values to capture the same amount of variance captured in the first two singular values of the average rating training matrix. Bar charts depicting these stored variances of both the MovieLens 100K and 1M data sets can be found in Figure 3.2 and Figure 3.3 respectively.

The goal of these experiments was to find the optimal parameters for our two machine learning

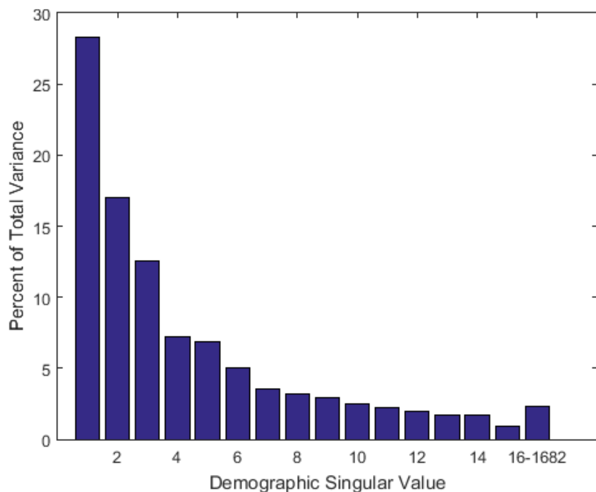


(a) MovieLens 100K: Variance in Training Ratings

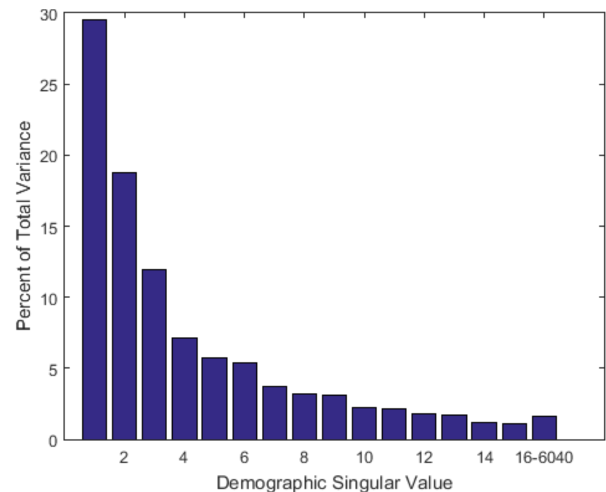


(b) MovieLens 1M: Variance in Training Ratings

Figure 3.2: The average total variance captured by the singular values of the training rating data. The average was taken of the five training rating sets "u1_base" through "u5_base" and "u1_base_1m" through "u5_base_1m" respectively. The sum of the variance captured in all singular values past 10 are shown in the last bar for each chart.



(a) MovieLens 100K: Variance in Item Demographics



(b) MovieLens 1M: Variance in Item Demographics

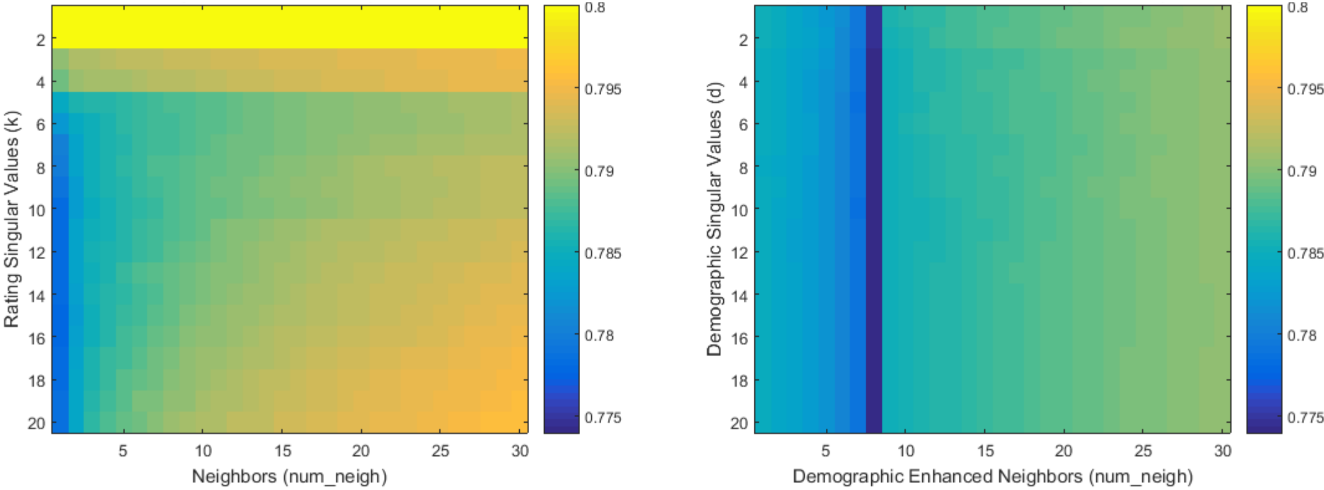
Figure 3.3: The total variance captured by the demographic singular values for each MovieLens data set tested. The sum of the variance captured in all singular values past 15 are shown in the last bar for each chart.

recommendation algorithms (ItemSvdRec and ItemSvdDemoRec) that optimized one of our evaluation metrics. The evaluation metric selected was Mean Absolute Error (MAE). We also wanted to view how the algorithms performed given the found optimized parameters for MAE under various other evaluation metrics. The three other metrics tested were Root Mean Squared Error (RMSE), Precision, and Recall. All of these evaluation metrics are described in detail in Section 2.4.

For ItemSvdRec the parameters needed are the number of singular values (k) used to make a kth approximation of the training matrix, and the number of neighbors (num_neigh) to formulate neighbor-

hoods around each item. To visualize how much k and num_neigh affected various evaluation metrics, we decided to display this three-dimensional data into a two-dimensional surface. For each plot the x-axis represents the number of neighbors ranging from 1 to 30, and the y-axis represents the number of used rating singular values ranging from 1 to 20. Each pixel of the image represents the third dimensional evaluation metric. These scales differ by range for each evaluation metric in order to capture the nuances of each metric.

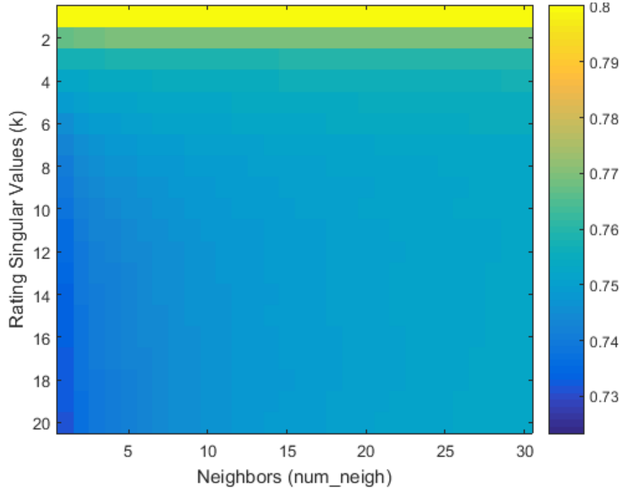
For ItemSvdDemoRec, k and num_neigh are also parameters, but there is an addition of the number of singular values (d) used to make a dth approximation of the demographic matrix. We decided to again utilize a two-dimensional surface for visualization. Since ItemSvdDemoRec differs from ItemSvdRec by additional item demographics used in neighborhood formation, we fixed k to be the optimal value found in our tests on ItemSvdRec that maximized the MAE. For these plots the x-axis represents the number of neighbors ranging from 1 to 30, and the y-axis represents the number of used demographic singular values ranging from 1 to 20. Each pixel of the image once again represents the third dimensional evaluation metric. In total this provides 600 data points or pixels for each individual plot. The remainder of this section is the collection of all evaluation plots created and their descriptions.



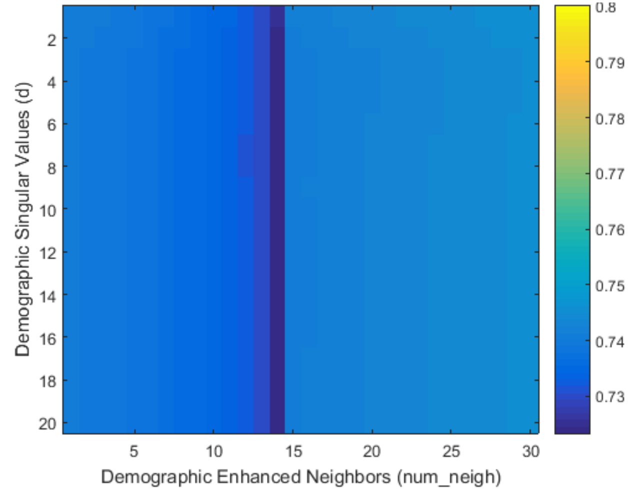
(a) MovieLens 100K: MAE of ItemSvdRec

(b) MovieLens 100K: MAE of ItemSvdDemoRec

Figure 3.4: The average Mean Absolute Error (MAE) of predictions generated for five disjoint rating testing subsets of the MovieLens 100K data set. Overall, lower MAE values (darker) indicate better performance. **(a)** Each pixel represents the MAE for each pair of parameters (num_neigh, k) for ItemSvdRec. Optimal parameters of $(1, 15)$ provided an MAE of 0.7780. **(b)** Each pixel represents the MAE for each parameter pair (num_neigh, d) for ItemSvdDemoRec with a fixed k value of 15, the optimal k value for MAE found in (a). Optimal parameters of $(8, 20)$ provided an MAE of 0.7740.

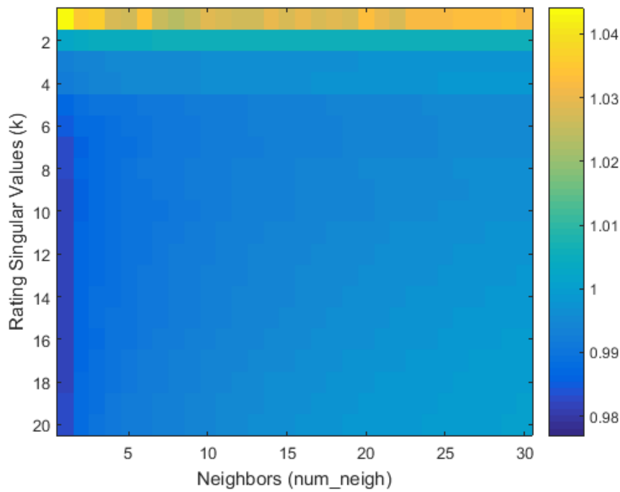


(a) MovieLens 1M: MAE of ItemSvdRec

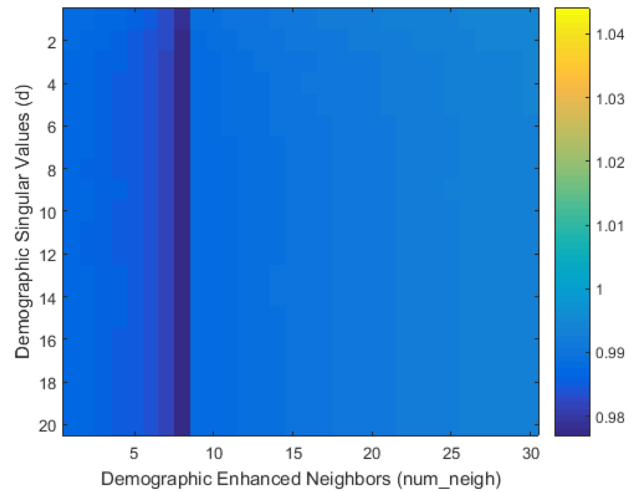


(b) MovieLens 1M: MAE of ItemSvdDemoRec

Figure 3.5: The average Mean Absolute Error (MAE) of predictions generated for five disjoint rating testing subsets of the MovieLens 1M data set. Overall, lower MAE values (darker) indicate better performance. **(a)** Each pixel represents the MAE for each pair of parameters (num_neigh, k) for ItemSvdRec over a 1 to 5 integer rating scale. Optimal parameters of $(1, 20)$ provided an MAE of 0.7317. **(b)** Each pixel represents the MAE for each parameter pair (num_neigh, d) for ItemSvdDemoRec with a fixed k value of 20, the optimal k value for MAE found in (a). Optimal parameters of $(14, 20)$ provided an MAE of 0.7234.

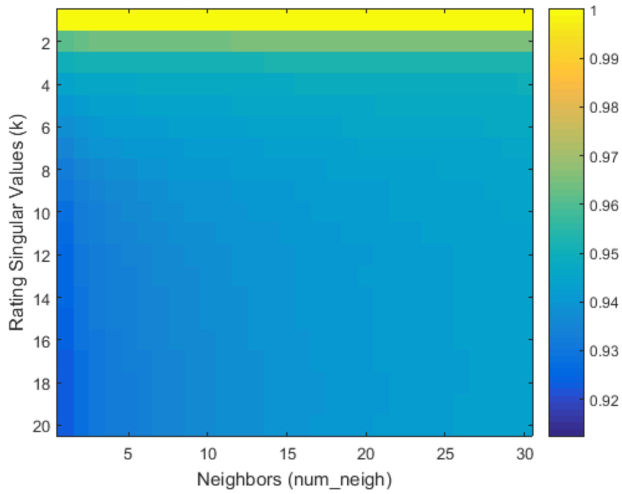


(a) MovieLens 100K: RMSE of ItemSvdRec

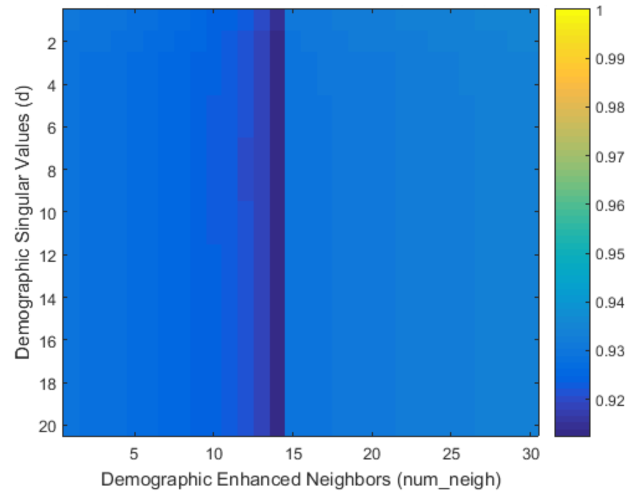


(b) MovieLens 100K: RMSE of ItemSvdDemoRec

Figure 3.6: The average Root Mean Squared Error (RMSE) of predictions generated for five disjoint rating testing subsets of the MovieLens 100K data set. Overall, lower RMSE values (darker) indicate better performance. **(a)** Each pixel represents the RMSE for each pair of parameters (num_neigh, k) for ItemSvdRec over a 1 to 5 integer rating scale. Optimal parameters of $(1, 10)$ provided an RMSE of 0.9815. **(b)** Each pixel represents the RMSE for each parameter pair (num_neigh, d) with a fixed k value of 15, the optimal k value for MAE found in (a) of Figure 3.4. Optimal parameters of $(8, 20)$ give an RMSE of 0.9814.

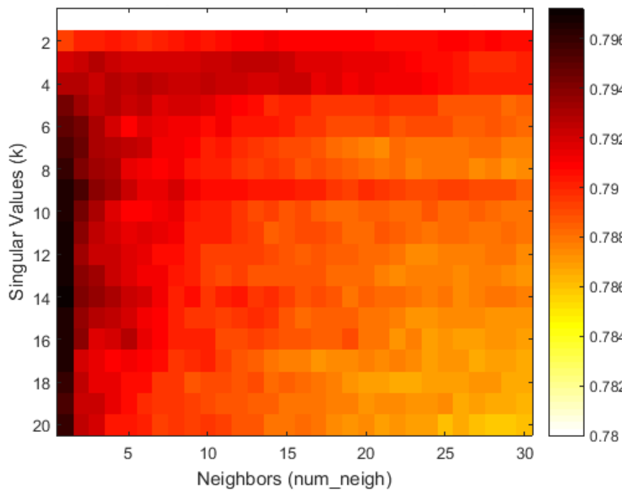


(a) MovieLens 1M: RMSE of ItemSvdRec

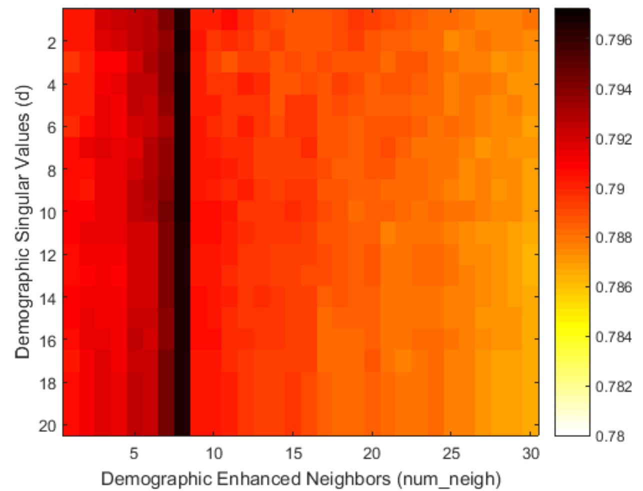


(b) MovieLens 1M: RMSE of ItemSvdDemoRec

Figure 3.7: The average Root Mean Squared Error (RMSE) of predictions generated for five disjoint rating testing subsets of the MovieLens 1M data set. Overall, lower RMSE values (darker) indicate better performance. **(a)** Each pixel represents the RMSE for each pair of parameters (num_neigh, k) for ItemSvdRec. Optimal parameters of $(1, 20)$ gave an RMSE of 0.9223. **(b)** Each pixel represents the RMSE for each parameter pair (num_neigh, d) with a fixed k value of 20, the optimal k value for MAE found in (a) of Figure 3.5. Optimal parameters of $(14, 20)$ gave an RMSE of 0.9125.

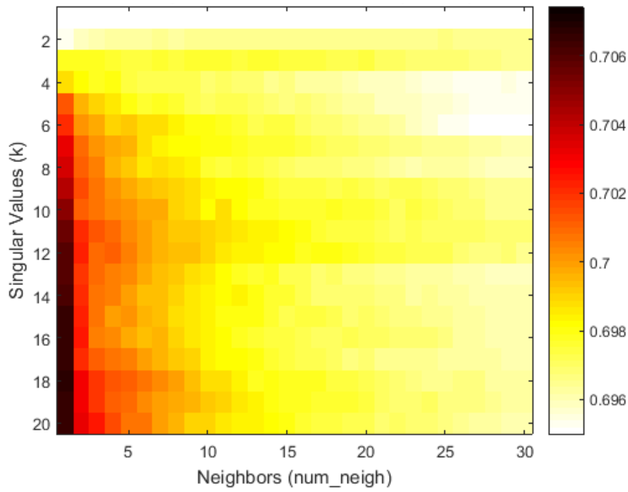


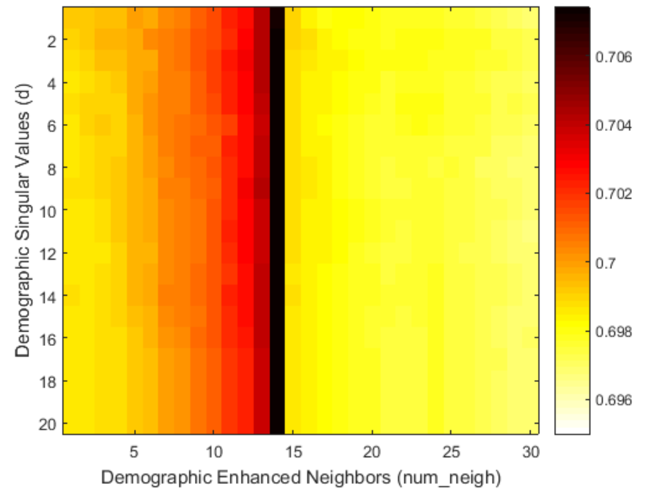
(a) MovieLens 100K: Precision of ItemSvdRec



(b) MovieLens 100K: Precision of ItemSvdDemoRec

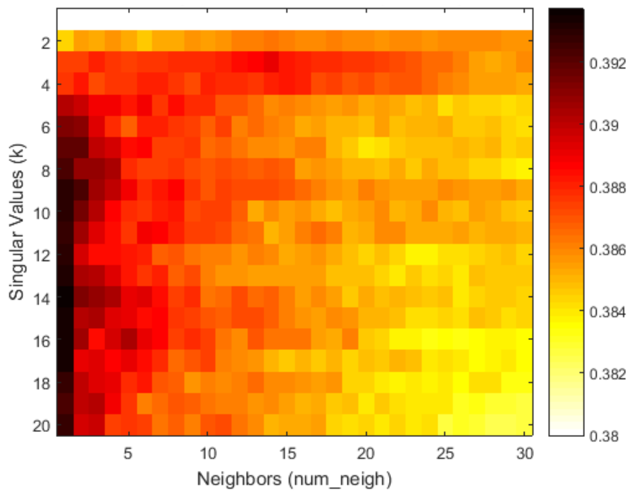
Figure 3.8: The average Precision of the rating prediction rankings generated for five disjoint testing subsets of the MovieLens 100K data set. Overall, higher Precision ranking values (darker) indicate better performance. **(a)** Each pixel represents the Precision for each pair of parameters (num_neigh, k) for ItemSvdRec. Optimal parameters of $(1, 14)$ provide a Precision of 0.7972. **(b)** Each pixel represents the Precision for each parameter pair (num_neigh, d) with a fixed k value of 15, the optimal k value for MAE found in (a) of Figure 3.4. Optimal parameters of $(8, 20)$ gave a Precision of 0.7967.

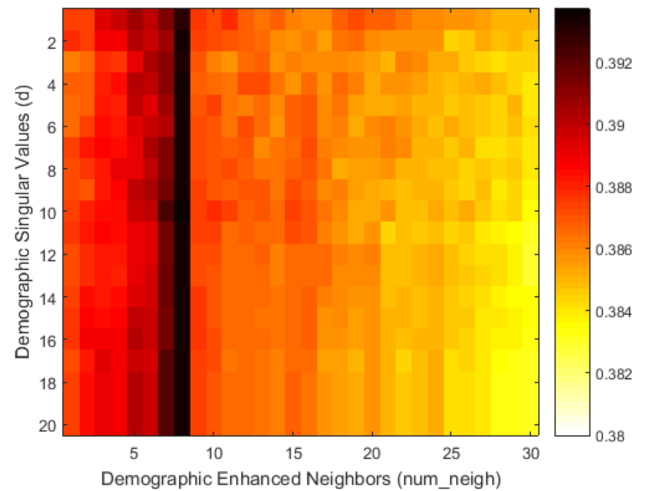




(a) MovieLens 1M: Precision of ItemSvdRec **(b)** MovieLens 1M: Precision of ItemSvdDemoRec

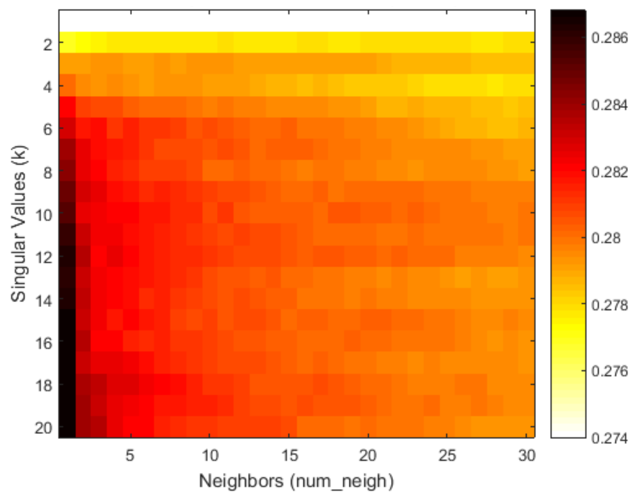
Figure 3.9: The average Precision of the rating prediction rankings generated for five disjoint testing subsets of the MovieLens 1M data set. Overall, higher Precision ranking values (darker) indicate better performance. **(a)** Each pixel represents the Precision for each pair of parameters (num_neigh, k) for ItemSvdRec. Optimal parameters of $(1, 18)$ gave a Precision of 0.7067. **(b)** Each pixel represents the Precision for each parameter pair (num_neigh, d) with a fixed k value of 20, the optimal k value for MAE found in (a) of Figure 3.5. Optimal parameters of $(14, 20)$ gave a Precision of 0.7074.



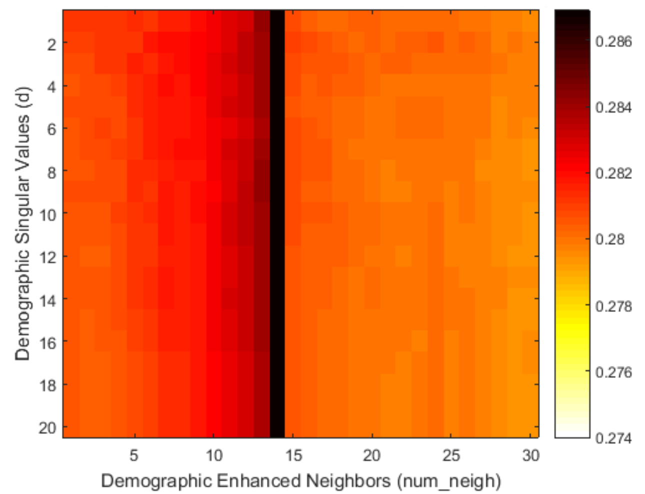


(a) MovieLens 100K: Recall of ItemSvdRec **(b)** MovieLens 100K: Recall of ItemSvdDemoRec

Figure 3.10: The average Recall of the rating prediction rankings generated for five disjoint testing subsets of the MovieLens 100K data set. Overall, higher Recall ranking values (darker) indicate better performance. **(a)** Each pixel represents the Recall for each pair of parameters (num_neigh, k) for ItemSvdRec. Optimal parameters of $(1, 14)$ gave a Recall of 0.3937. **(b)** Each pixel represents the Recall for each parameter pair (num_neigh, d) with a fixed k value of 15, the optimal k value for MAE found in (a) of Figure 3.4. Optimal parameters of $(8, 10)$ gave a Recall of 0.3935.



(a) MovieLens 1M: Recall of ItemSvdRec



(b) MovieLens 1M: Recall of ItemSvdDemoRec

Figure 3.11: The average Recall of the rating prediction rankings generated for five disjoint testing subsets of the MovieLens 1M data set. Overall, higher Recall ranking values (darker) indicate better performance. **(a)** Each pixel represents the Recall for each pair of parameters (num_neigh, k) for ItemSvdRec. Optimal parameters of $(1, 20)$ gave a Recall of 0.2868. **(b)** Each pixel represents the Recall for each parameter pair (num_neigh, d) with a fixed k value of 20, the optimal k value for MAE found in (a) of Figure 3.5. Optimal parameters of $(14, 20)$ gave a Recall of 0.2870.

Chapter 4

Discussion

The prediction evaluations over the MovieLens's data sets will be discussed from two major perspectives. The first involves comparisons between the performances of the two algorithms, ItemSvdRec and ItemSvdDemoRec. How the inclusion or exclusion of item demographics in a recommendation system affects its prediction accuracy can help lead to better the overall performance of the system. The second perspective involves how the data sets themselves affect system performance. As described in Section 3.2, the MovieLens 100K and MovieLens 1M data sets are similar in terms of sparsity. However, MovieLens 1M has roughly six times more users and there are over twice the number of movies available for recommendation than that of MovieLens 100K. How the algorithms behave when sparsity is preserved, but operation occurs over a larger scale is yet another interesting investigation. These perspectives are intertwined and findings from one view help to understand the other. In addition to these investigations, an additional section discussing results from an alternative data set, Jester 2, is provided. We wished to discuss this alternative data to reflect that these algorithms can operate over data sets with continuous and varying rating scale values.

4.1 Performance over MovieLens Data Sets

One must first discuss the parameter tuning of the algorithms. From the average evaluation metric results in Section 3.3, one clearly notices that the optimal number of neighbors (num_neigh) for all metrics is 1 under ItemSvdRec. As we increased the number of neighbors, all metrics steadily became sub-optimal. Plot (a) from Figures 3.4 to 3.11 indicate that when item neighbors are solely generated by similarity to other items in terms of past ratings, it is optimal for each item to have only themselves as neighbors. In other words, to predict the ratings of an item, ItemSvdRec does not find it useful take into account the adjusted cosine similarity between other items. As such, unless specified, the following assumes a fixed num_neigh value for ItemSvdRec of 1.

Mean Absolute Error (MAE) was selected to be the primary metric to optimize in our research, and as such, it will be the main metric focused on in the following paragraphs. The results for ItemSvdDemoRec were obtained using the optimal number of rating singular values (k) as found through parameter tuning when optimizing the MAE evaluation of ItemSvdRec. These k values showed to be slightly different then those which optimize other performance metrics. Table 4.1 provides the evaluation metrics of ItemSvdRec over MovieLens 100K for their individual optimal k value as well as for the optimal MAE k value. The results showed that optimizing the system for one evaluation metric, overall system performance for each other evaluation metric decreased. Recall decreased the most using a k

equal to 14 rather than 15 with a -0.10% overall improvement. This change in performance may seem small to some, and they would be correct. Essentially, if one wishes to optimize the recommendation system for MAE, they may do so without severely compromising other metrics. However, with the many different factors that go into recommendation systems, even small improvements can be significant. For illustration, the difference between the top two and third place team's algorithms in the "Netflix Prize" one-million dollar competition in terms of percent improvement by Root Mean Squared Error (RMSE) over Cinematch was 0.16% [1]. It should also be reminded that although experimental errors do exist in this and all other evaluation techniques, the five fold cross evaluation serves to help minimize these types of errors.

Evaluation Metric	**Optimal k: Metric**	**Evaluation**	**Optimal k: MAE**	**Evaluation**	**Percent Improvement**
MAE	15	0.7780	15	0.7780	0.00%
RMSE	10	0.9815	15	0.9819	−0.04%
Precision	14	0.7972	15	0.7966	−0.08%
Recall	14	0.3937	15	0.3933	−0.10%

Table 4.1: Comparison of average prediction evaluation metrics over the MovieLens 100K data set from the ItemSvdRec algorithm. This table's values reflect the change in improvement from using the number of rating singular values, k, that optimizes MAE rather than the k values that optimize each alternative evaluation metric. The number of neighbors parameter, num_neigh, was fixed at one. When compounded upon one another, small pertubations in percent change, like the ones shown, can lead to a significant change in overall recommendation system success.

In each of the (a) plots from Figure 3.4 to 3.11, we see that having too few singular values results in very poor prediction accuracy metrics for ItemSvdRec. In both the MovieLens 100K and 1M data sets, once the variance stored in the first two singular values are encompassed into the creation of pseudo-users in ItemSvdRec, increasing k leads to a much smaller yet still important increase in performance. This is supported by our findings from the Principal Component Analysis (PCA) utilizing Singular Value Decomposition (SVD) to find total average variance captured in the singular values of these data sets. Figure 3.2 depicts that roughly 96 and 99 percent of the total variance in the MovieLens 100K and 1M data sets respectively are stored within the first two singular values. By setting k equal to 2, an MAE of 0.7996 is produced over MovieLens 100K. Doing the same over MovieLens 1M provides an MAE of 0.7672. The optimal k values for these two data sets of 15 and 20 respectively give a much lower MAE of 0.7780 and 0.7317. This can be seen to occur with RMSE, Precision, and Recall as well. This implies that even though the first two singular values capture almost the entirety of the variance, it is still not enough to achieve optimal system performance. However, this does not imply simply increasing k indefinitely leads to better performance. Within our tested parameter ranges, increasing k past 15 actually increased MAE over MoveLens 100K. A k value of 20 produced a MAE of 0.7786, a small yet measurable increase. The optimal k value for MovieLens 1M was 20, and a similar trend would have likely been seen if the testing parameter range for this data set was increased. In short, it is difficult to find the exact optimal number of singular values without performing prediction evaluation, and that variance captured is not an exact indicator of system performance.

Two observations are apparent from the collection of (b) plots from Figures 3.4 to 3.11. First is that the number of singular values used in the SVD of the item demographic matrix in ItemSvdDemoRec does not significantly impact performance. Although Figure 3.4 (b) and Figure 3.5 (b) indicate a small performance increase after setting the number of demographic singular values, d, to 3, any additional increase in d results in stagnant MAE performance. The second observation is that ItemSvdDemoRec actually utilizes some number of neighbors when optimized. This version of *num_neigh* not only takes into consideration the similarity between items based on past ratings, but is also enhanced through item similarity in terms of item demographics through a Pearson correlation. ItemSvdDemoRec is optimal for all tested prediction metrics with a neighborhood of 8 over MovieLens 100K, and a neighborhood of 14 over MovieLens 1M. This indicates that the combination of using item rating histories and demographic information in neighborhood formulation works better than rating histories alone. MAE over MovieLens 100K dropped from a previously optimal 0.7780 in ItemSvdRec to 0.7740 in ItemSvdDemoRec, a 0.51% improvement. Similarly, MAE over MovieLens 1M decreased from a previously optimal 0.7317 to 0.7234, a 1.13% improvement.

As mentioned previously, RMSE, Precision, and Recall all behaved similarly as parameter tuning was done to minimize MAE. There were, however, some key differences that should be noted. MAE and RMSE were the two rating prediction measures used to evaluate the algorithms. The average RMSE was much greater than the MAE for every parameter set tested, which is common for movie recommendation systems. The best RMSE achieved over the MovieLens 100K data set was 0.9815 through ItemSvdRec and 0.9814 through ItemSvdDemoRec as can be seen in Figure 3.6. Comparatively, the MovieLens 1M data set had a best RMSE of 0.9223 through ItemSvdRec and 0.9125 through ItemSvdDemoRec as can be seen in Figure 3.7. It is interesting that the larger of the two data sets had a much more significant improvement by including item demographics. The reason for this may be two fold. First, the range tested for the parameter k may have been too small for the parameter tuning over MovieLens 1M. Table 4.1 shows that, for MovieLens 100K, the optimal k for MAE was much greater than the optimal k for RMSE. This is likely the same case for MovieLens 1M. Due to the fact that the optimal k for MAE and RMSE happen to both be 20, the upper extrema of our testing range, we may have used a k closer to the optimal k for RMSE than MAE. Second, MovieLens 1M has over twice the number of items and item demographics available over MovieLens 100K. Since there are more movies that are similar in terms of ratings and demographics, better and larger neighborhoods are formed using this data set.

Precision and Recall were the two usage or ranking measures used to evaluate the two algorithms. As can be seen in Figures 3.8 to 3.11, optimal Precision and Recall between algorithms remained almost unchanged. This does not necessarily reflect negatively on the addition of item demographics. As the k used in ItemSvdDemoRec was not optimized for either Precision nor Recall, it is surprising that these measures did not decrease. Both algorithms performed significantly better in Precision than in Recall over each data set. Precision was consistently within the score range of 0.68 to 0.80, while Recall consistently had scores between 0.27 and 0.40. Typically, Precision is weighted more important than Recall. Thus, these algorithms have a favorable Precision to Recall ratio. However, if one wanted to value Recall more, then the number of movies ranked for each user would have to increase significantly.

Data Set	Algorithm	MAE	NMAE	RMSE	NRMSE	Precision	Recall
MovieLens 100K	ItemSvdRec	0.7780	0.1945	0.9815	0.2454	0.7972	0.3937
	ItemSvdDemoRec	0.7740	0.1935	0.9814	0.2454	0.7967	0.3935
MovieLens 1M	ItemSvdRec	0.7317	0.1829	0.9223	0.2306	0.7067	0.2868
	ItemSvdDemoRec	0.7234	0.1809	0.9125	0.2281	0.7074	0.2870
Jester 2	ItemSvdRec	3.475	0.1737	4.405	0.2203	0.9500	0.7625

Table 4.2: The optimal evaluation metrics found for each data set and algorithm pair tested. Since the MovieLens and Jester data sets have different scales for rating, Normalized MAE (NMAE) and Normalized RMSE (NRMSE) are provided for ease of comparison. Overall, ItemSvdDemoRec outperforms ItemSvdRec over the MovieLens data sets. ItemSvdRec over Jester 2 outperforms all metrics from either algorithm over the MovieLens data sets.

4.2 Alternative Data Set: Jester Online Joke Recommender

We also wanted to express that these algorithms, and the mathematical techniques within them, can operate over alternative data sets outside that of the common five star movie rating system. As such, we ran ItemSvdRec over one of the publicly available data sets from Jester, the online joke recommending system. Jester has collected millions of joke ratings from thousands of users, and has been used in several prominent papers on collaborative filtering algorithms [17]. We used Jester Dataset 2, which has roughly 1.7 million ratings from 63,978 users over 150 jokes. Rather than a 1 to 5 integer rating scale, this data set has continuous ratings between -10.00 and 10.00. This data set does not include joke demographics, and thus ItemSvdDemoRec could not be ran using this data. In addition, pre-processing had be done first in order to manipulate the data into matrix form as well as create five fold cross validation training and testing subsets. This process was similar to that described in Section 3.2.

Once again in Figure 4.1, we see that the first two singular vectors encompass the majority of the variation in the data. This time, roughly 82.5% of the data is stored here. This however is not enough variation captured for optimal system performance. MAE and RMSE both attain their minimums with a k value of 4 and a *num_neigh* value of 1. Precision and Recall attain their maximums with a k value of 20 and a *num_neigh* of 1. Since the MovieLens and Jester data sets have different ratings scales, we normalized the best MAE and RMSE of each data set and algorithm. These values are given in Table 4.2. One can see that ItemSvdRec over Jester 2 achieved the best performance in all categories, even outperforming ItemSvdDemoRec over the MovieLens data sets. This is because Jester 2 has a higher user to movie ratio as well as more known ratings than either of the MovieLens data sets. For comparison, Jester 2 had a sparsity of about 17.7% while MovieLens 100K and 1M had sparsities of 6.3% and 4.2% respectively. In addition, the usage metrics over Jester 2 are large because there are only 150 total jokes to recommend to any given user. The algorithms must operate over thousands of items in the MovieLens data sets, so they must predict the top ten movies from a much larger set of possible choices.

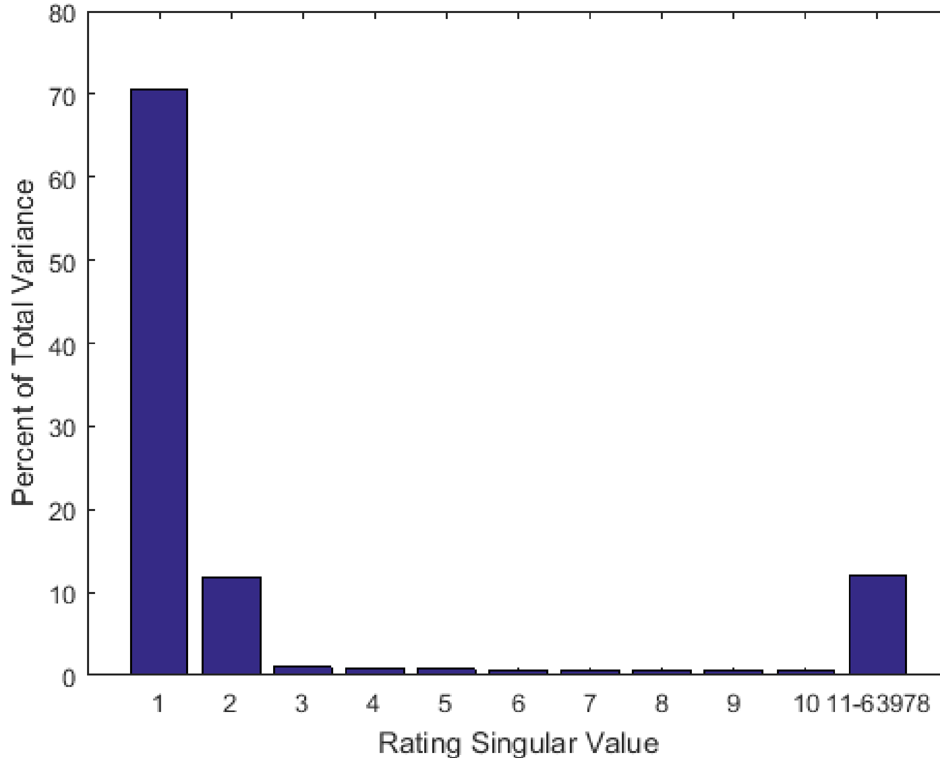
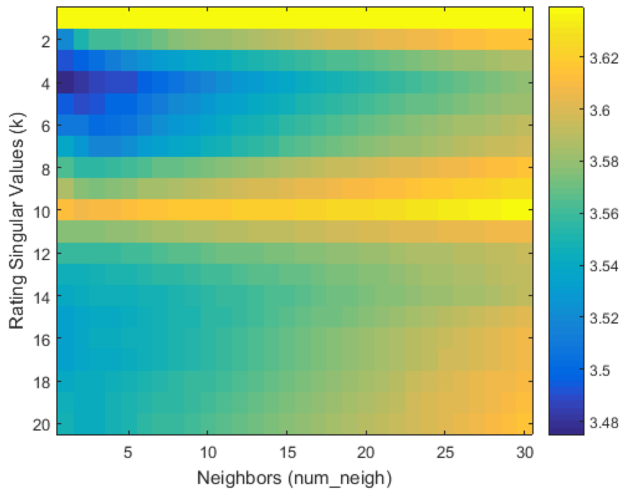


Figure 4.1: Average total variance captured by the singular values of training rating subsets from the Jester 2 data set. The first two singular values encompass on average roughly 82.5% of the data. The sum of the variance captured in all singular values past 10 are shown in the last bar of the chart.

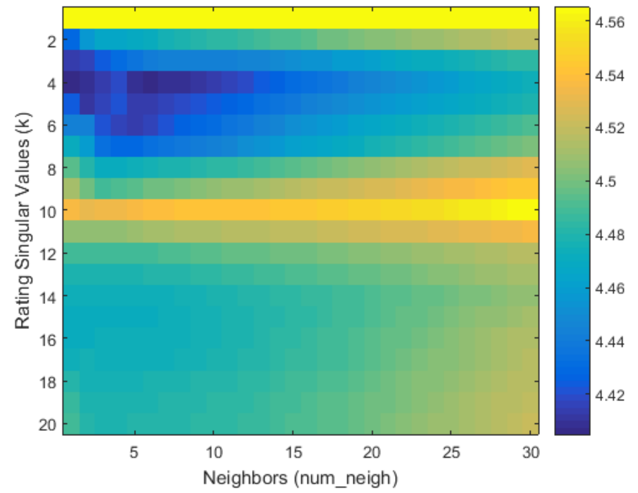
4.3 Conclusion and Future Research

Machine learning is a vast field. It encompasses several different disciplines including mathematics, statistics, and computer science. It does not provide an answer to every data driven question, but it rather gives us the ability to explore and understand the data we produce and collect like never before. Whether these learning algorithms are supervised, unsupervised, or reinforcement driven, they each help us make data-driven decisions and predictions. Recommendation systems are one small yet meaningful application of machine learning. They provide a positive feedback loop in the recommendation process. The better the recommendation system, the more users utilize it. The more users, the more data is given to the system to then learn from. The system internalizes that data and the process repeats itself. Mathematical techniques such as PCA, SVD, and neighborhood formation are just a few examples of how many of these systems process this plethora of data in a manageable time frame. Rating evaluation methods, such as MAE or RMSE, and usage prediction methods, such as Precision and Recall, serve as a way to gauge the learning performance of the system. Whether we wish to recommend something as simple as a movie or joke, or something as complex as a treatment or medication to someone, machine learning and recommendation systems are powerful ideas that can better the individual and in turn society as a whole.

This paper just scratches the surface of both machine learning and recommendation systems. Future work can, and is, being done in both these domains. In terms of this paper, one could extend upon both ItemSvdRec and ItemSvdDemoRec to utilize user demographics, time stamps, and other

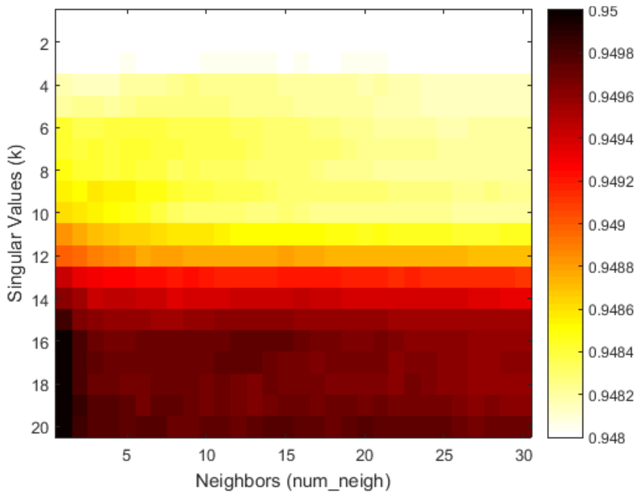


(a) Jester 2: MAE Values

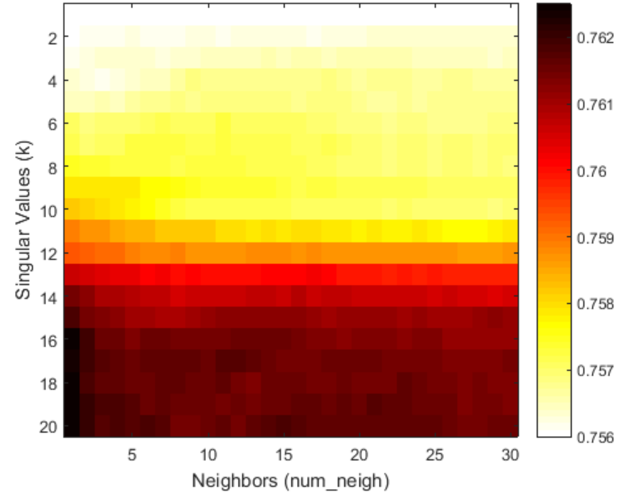


(b) Jester 2: RMSE Values

Figure 4.2: The average rating prediction metrics, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), produced from the predictions generated by ItemSvdRec over five disjoint rating testing subsets of the Jester 2 joke data set. Each pixel represents the metric for each parameter pair (*num_neigh*, *k*) using a -10.00 to 10.00 continuous rating scale. Overall, lower MAE and RMSE values (darker) indicate better performance. **(a)** Optimal parameters of (1,4) provided an MAE of 3.475. **(b)** Optimal parameters of (1,4) provided an RMSE of 4.405.



(a) Jester 2: Precision Values



(b) Jester 2: Recall Values

Figure 4.3: The average usage prediction metrics, Precision and Recall, produced from the predictions generated by ItemSvdRec over five disjoint rating testing subsets of the Jester 2 joke data set. Each pixel represents the metric for each parameter pair (*num_neigh*, *k*). Overall, greater Precision and Recall values (darker) indicate better performance. **(a)** Optimal parameters of (1,20) provided an Precision of 0.9500. **(b)** Optimal parameters of (1,20) provided an Recall of 0.7625.

additional information provided in the MovieLens data sets. Alternative data sets could also be studied such as the Book-Crossing Data Set found in *Improving Recommendation Lists Through Topic Diversification* [48]. Individuals with a strong background in computer science could generate their own data through web crawling and other data extraction techniques on websites such as Twitter. Those with a strong mathematical or statistical background could attempt alternative matrix factorization and

neighborhood formation techniques as well.

Appendix A

MATLAB Code

A.1 Item Filtering with SVD Recommender (ItemSvdRec)

```
function [ratings_predicted] = item_SVD_recommender(ratings, k, num_neigh)
%% ItemSvdRec An algorithm that uses item-based filtering enhanced by SVD.
% A mxn ratings matrix (ratings) with m-users and n-items.
% Preprocessing eliminates all missing data (100% coverage) using user-item
% averages (ratings_norm). The k-th approximation SVD of the filled matrix
% is taken (ratings_reduced), and both user and
% item characterization matrices (pu and qi) are created. Neighborhood
% formation of items is done through Adjusted Cosine Similarity, and final
% prediction generation (ratings_predicted) is achieved.
% The pseudo-code for the algorithim outline can be found in:
% Applying SVD on Generalized Item-based Filtering
%                   - (http://www.tmrfindia.org/ijcsa/V3I34.pdf)

%% Parameters
% ratings : A user-by-item ratings matrix. Ratings can be any value, but
%           null values must be of the form NaN.
% k :  Integer <= rank(ratings). Gives kth-approximation of SVD.
% num_neigh: Number <= size(ratings,2) i.e. number of total items.
%            When finding similar neighbors, only the first num_neigh
%            neighbors will be considered similar enough.

%% Item-based Filtering Enhanced by SVD (ItemSvdRec)
%% 1: Define user-item matrix ratings
[m,n] = size(ratings);  % m is number of users, n is number of items

%% 2: Preprocessing (solving sparsity problem)
% 2(a): Compute row (user) average and column (item) average.
row_avg = nanmean(ratings,2);   % size 1-by-m, computes average of all rows/users
row_avg(isnan(row_avg))= nanmean(row_avg);     % if a user did not rate any movie set
                                               %  to the overall user average
col_avg = nanmean(ratings,1);   % size 1-by-n, computes average of all columns/items
col_avg(isnan(col_avg))= nanmean(col_avg);  % if a movie was not rated by any user set
                                            % to the overall item average

% 2(b): Replace all entries with missing values (NaN) with corresponding col_avg
ratings_norm = ratings;      % initialize normalized ratings matrix
for i = 1:n                  % loops through all columns and replaces all
                             %  NaN with corresponding columns average
```

```
39      ratings_norm(isnan(ratings_norm(:,i)),i) = col_avg(i);
    end
41
    % 2(c): Subtracts corresponding row average from all entries
43  % bsxfun applies element-by-element operations on two arrays
    ratings_norm = bsxfun(@minus,ratings_norm, row_avg);
45
    %% 3 & 4: Compute the SVD of ratings_norm and obtain k-th approximation
47  [Uk,Sk,Vk] = svds(ratings_norm, k); % rank k SVD approximation
    ratings_reduced = Uk*Sk*Vk';         % reduced k-approx ratings matrix
49
    %% 5: Compute user and item characteristic vectors, pu and qi
51  pu = Uk*sqrt(Sk)';    % size m-by-k, user associated vector of characterizations
    qi = sqrt(Sk)*Vk';    % size k-by-n, item associated vector of characteristics
53
    %% 6: Neighborhood Formation (similar item neighbors)
55  % 6(a): Adjusted Cosine Similarity (NOTE: This differs from plain adjusted
    % cosine similarity as the bias in different rating scales was taken into
57  % cosideration when normal-ization to create ratings_norm)
        sim_matrix = zeros(n,n); % initialize similarity matrix
59      for j=1:n % loop through all items
            %size 1-by-n, summation of product of item j ratings and each other
61          %   item ratings
            numerator = sum(bsxfun(@times, qi(:,j), qi),1);
63          %size 1-by-n, normalize
            denominator = sqrt(sum(qi(:,j).^2,1).*sum(qi.^2,1));
65          sim_matrix(j,:) = numerator./denominator; % update similarity matrix
        end
67
    %% 6(b) & 7: Isolate most similar items & Prediction Generation
69  ratings_predicted = zeros(m,n); % initialize predicted ratings matrix

71  for j=1:n %loop through all items
        % sort similar items to item j, and find their indices
73      [~, sort_itemj_indices] = sort(sim_matrix(:,j),'descend');
        % best num_neigh neighbors to item j
75      sel_items = sort_itemj_indices(1:num_neigh);

77      % Add user row average to item j neighbors from reduced ratings matrix,
        %  multiply by similarity of each neighbor, sum ratings up and divide
79      %  by total similar neighbors. The reduced ratings of the closest item
        %  neighbors with the user average added back in.
81      ratings_red_j_plus_row_avg = bsxfun(@plus,ratings_reduced(:,sel_items),row_avg);
        % similarity to closest neighbors
83      sim_vector_j = sim_matrix(j,sel_items);
        % sum of absolute values of similarity to item j
85      sim_vector_abs_sum = sum(abs(sim_vector_j));
        % predicted ratings for item j for all users
87      ratings_predicted(:,j) = sum(bsxfun(@times,ratings_red_j_plus_row_avg,...
            sim_vector_j),2)./sim_vector_abs_sum;
89  end
    end
```

A.2 Item Filtering with SVD and Item Demographics Recommender (ItemSvdDemoRec)

```
function [ratings_predicted] = item_SVD_demo_recommender(ratings, k, item_demographics, d, num_neigh)
%% ItemSvdDemoRec An algorihm that uses item-based filtering, item demographics, and SVD.
% A m by n ratings matrix (ratings) with m-users and n-items, and an x by n item
% demographics matrix (item_demographics) where x is the number of tested
% demographics are input. SVD is conducted on the item demographics with d
% singular values. Preprocessing eliminates all missing data (100% coverage)
% using user-item averages (ratings_norm) in the ratings matrix. The k-th
% approximation SVD of the filled matrix is taken (ratings_reduced), and both
% user and item characterization matrices (pu and qi) are created. Neighborhood
% formation of items is done through Adjusted Cosine Similarity of the items and
% the demographic correlation between each item. The final neighborhood similarity
% is the addition of the two similarities. The final prediction generation
% (ratings_predicted) is then achieved.
% The pseudo-code for the algorithim outline can be found in:
% Applying SVD on Generalized Item-based Filtering
%                   - (http://www.tmrfindia.org/ijcsa/V3I34.pdf)
%% Parameters
% ratings : A user-by-item ratings matrix. Ratings can be any value, but
%           null values must be NaN.
% k :  Integer <= rank(ratings). Gives kth-approximation of SVD for ratings matrix.
% item_demographics : A demographic-by-item demographics matrix. Each
%                     demographic must be either 0 (item does not have) or
%                     1 (item does have).
% d :  Integer <= rank(item_demographics). Gives dth-approximation of SVD
%      for the item_demographics matrix
% num_neigh: Number <= size(ratings,2) i.e. items. When finding similar
%            neighbors, only the first num_neigh neighbors will be considered
%            similar enough. This is the addition of the adjusted cosine
%            similarity and the demographic correlation of the items.


%% Item-based Filtering Enhanced by SVD and Item-Demographics
%% 1(a) : Construct demographic vectors for m users and n items and collect
%%        these demographic vectors in array
% item_demographics should be inputed in this form. See parameters.

%% 1(b)(case 2) : Perform SVD on Item Demographic matrix (item_demographics)
[Ud,Sd,Vd] = svds(item_demographics,d); % Rank d SVD approximation
qid = sqrt(Sd)*Vd'; % size d-by-n, item d latent demographic characteristic vectors

%% 2: Construct user-item ratings matrix of size m-by-n
% ratings should already be in this form. See parameters.
[m,n] = size(ratings);  % m is number of users, n is number of items

%% 3 (case 2): Neighborhood Formation
% 3(a) Preprocessing (fixing sparsity problem)
% : Compute row (user) average and column (item) average.
%      NOTE: Other preprocessing methods exist, this is one method
row_avg = nanmean(ratings,2);   % size 1-by-m, computes average of all rows/users
row_avg(isnan(row_avg))= nanmean(row_avg);    % if a user did not rate any movie set
```

```matlab
                                                   %  to the overall user average
col_avg = nanmean(ratings,1);   % size 1-by-n, computes average of all columns/items
col_avg(isnan(col_avg))= nanmean(col_avg);  % if a movie was not rated by any user set
                                            %   to the overal item average
% 3(b): Replace all matrix entries with no values (NaN) with corresponding col_avg
ratings_norm = ratings;     % initialize normalized ratings matrix
for i = 1:n                 % loops through all columns and replaces all
                            %   NaN with corresponding columns average
   ratings_norm(isnan(ratings_norm(:,i)),i) = col_avg(i);
end

% 3(c): Subtracts corresponding row average from all entries
% bsxfun applies element-by-element operations on two arrays
ratings_norm = bsxfun(@minus,ratings_norm, row_avg);

%% 3(d): Compute the SVD of ratings_norm and obtain k-th approximation
%k = 6;      % Size of dimensonality reduced space, matlab defaults to first
            %   6 singular values, increasing k (generally) betters
            %   accuracy but increases computational time
[Uk,Sk,Vk] = svds(ratings_norm, k); % rank k SVD approximation
ratings_reduced = Uk*Sk*Vk';        % reduced k-approx ratings matrix

%% 3(e): Compute user and item characteristic vectors, pu and qi
pu = Uk*sqrt(Sk)';    % size m-by-k, user associated vector of characterizations
qi = sqrt(Sk)*Vk';    % size k-by-n, item associated vector of characteristics

%% 3(f): Item Correlation (find similarity and item neighbors)
%   Adjusted Cosine Similarity (NOTE: This differs from plain adjusted
% cosine similarity as the bias in different rating scales was taken into
% cosideration when normal-ization to create ratings_norm)
    sim_matrix = zeros(n,n); % initialize similarity matrix
    for j=1:n % loop through all items
        % size 1-by-n, summation of product of item j ratings and each other
        %   item ratings
        numerator = sum(bsxfun(@times, qi(:,j), qi),1);
        %size 1-by-n, normalize
        denominator = sqrt(sum(qi(:,j).^2,1).*sum(qi.^2,1));
        sim_matrix(j,:) = numerator./denominator; % update similarity matrix
    end
%% 4 (case 2) : Calculate Demographic Correlation
%               (find similarity of psuedo item demographic vectors)
    dem_cor_matrix = zeros(n,n);
    for x=1:n % loop through all items
        % computes dot product of current column vec and all other column vecs
        numerator = sum(bsxfun(@times,qid(:,x), qid),1);
        denominator = norm(qid).*sqrt(sum(qid.^2,1)); % finds L2 norm of each
                                                      % column vector
        dem_cor_matrix(x,:) = numerator./denominator;
    end

%% 5 : Calculate the Enhanced Correlation
%      (based from both sim_matrix and dem_cor_matrix)
%      NOTE: Chose Enhanced Correlation to be of the form ratings similarity +
%             demographic correlation. There exists other forms of weighing.
    enhanced_cor = sim_matrix + dem_cor_matrix;

```

```
%% 6(b) & 7: Isolate most similar items & Prediction Generation
ratings_predicted = zeros(m,n); % initialize predicted ratings matrix

for j=1:n %loop through all items
    % sort similar items to item j, and find their indices
    [~, sort_itemj_indices] = sort(enhanced_cor(:,j),'descend');
    sel_items = sort_itemj_indices(1:num_neigh); % best num_neigh neighbors

    % Add user row average to item j neighbors from reduced ratings matrix,
    %  multiply by similarity of each neighbor, sum ratings up and divide
    %  by total similar neighbors. The reduced ratings of the closest item
    %  neighbors with the user average added back in.
   enh_corr_sel_item_vec = enhanced_cor(j,sel_items);
   ratings_reduced_plus_rowavg = bsxfun(@plus,ratings_reduced(:, sel_items),row_avg);

   ratings_predicted(:,j) = sum(bsxfun(@times, ratings_reduced_plus_rowavg, ...
       enh_corr_sel_item_vec),2)./sum(abs(enh_corr_sel_item_vec));
end
end
```

A.3 Data to Matrix: MovieLens 100K Example

```
%% load_ml_100k_dataset_ratings Overview:
%  Loads in MovieLens 100k Dataset of ratings for movies and creates a ratings matrix
%  'ml_100k_dataset_ratings'. This ratings matrix is users-by-items with a
%  null rating described by NaN.
%  ml-100k (5MB) contains 100,000 integer ratings between 1 and 5 on
%  943 users and 1682 movies.

%% Acknowledgment of source:
%  MovieLens data sets were collected by the GroupLens Research Project
%  at the University of Minnesota.
% URL: http://grouplens.org/datasets/movielens/
% Paper: F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets:
%        History and Context. ACM Trans. Interact. Intell. Syst. 5, 4, Article
%        19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

%% load_ml_100k_dataset_ratings
data_dir = 'ml-100k';

if exist(data_dir,'dir') ~= 7 % download and unzip dataset if directory
                              %  folder if does not exist
    unzip('http://files.grouplens.org/datasets/movielens/ml-100k.zip')
end

if ~exist('ml_100k_dataset_ratings','var') % load dataset if does not exist
data = readtable(fullfile(data_dir,'u.data'),'FileType','text',...
    'ReadVariableNames',false,'Format','%f%f%f%f');
data.Properties.VariableNames = {'user_id','movie_id','rating','timestamp'};

% create ratings matrix
ml_100k_dataset_ratings = accumarray([data.user_id, data.movie_id],...
```

```
    data.rating,[],[],NaN);

clear data ml_100k_demographics
end
```

A.4 5 Fold Cross Validation Data Sets Creation

```
%% createTrainingData Overview
% This function takes in a ratings matrix and creates training and testing
% subsets for a five fold cross validation. A random permutation of all
% rated indices is created and separated into five intervals of equal
% length. These intervals are then used to populate NaN matrices with the
% ratings in each interval from the ratings matrix. The test and training
% data is stored in a m-by-n-by-5 multidimensional array where m and n are
% the dimensions of the ratings matrix.
%% Parameters
% ratings:  An m-by-n ratings matrix. Entries with no ratings must contain
%           NaN.

%% createTrainingData
function [u_test, u_training] = createTrainingData(ratings)
    % initialize array to store all 5 test subsets
    u_test = NaN(size(ratings,1), size(ratings,2), 5);
    % initialize array to store all 5 training subsets
    u_training = NaN(size(ratings,1), size(ratings,2), 5);

    allIndices = find(~isnan(ratings)); % finds all indices that are rated
    randVec = allIndices(randperm(size(allIndices,1))); % randomly permutes indices
    intervalSize = floor(size(randVec,1)/5); % size of ~20% interval
    interval = 1:intervalSize; % 20% interval of indices

    for n = 1:5 % for each test/training set to be created
        % initialize temporary storage for test data
        u_test_temp = NaN(size(ratings));
        % initialize temporary storage for training data
        u_train_temp = ratings;

        % find current 20% interval of indices to use
        randVec20 = randVec(interval + (n-1)*intervalSize);
        % add ratings in 20% interval to testing matrix
        u_test_temp(randVec20) = ratings(randVec20);
        u_test(:,:,n) = u_test_temp;
        % remove all added ratings from training matrix (100-20=80%)
        u_train_temp(~isnan(u_test_temp)) = NaN;

        % store test and training data into multi-dimensional arrays for
        %   later access
        u_training(:,:,n) = u_train_temp;
        u_test(:,:,n) = u_test_temp;
    end
end
```

Bibliography

[1] *The Netflix Prize Rules*, 2006 (accessed 2016-02-05). http://www.netflixprize.com//rules.

[2] *LAPACK SGESVD SUBROUTINE*, 2011 (accessed 2016-01-16). http://www.netlib.org/lapack/single/sgesvd.f.

[3] Pierre Baldi and Søren Brunak. *Bioinformatics: The Machine Learning Approach*. Massachusetts Institute of Technology Press, 2001.

[4] John Barnard and Donald B Rubin. Small-sample Degrees of Freedom with Multiple Imputation. *Biometrika*, 86(4):948–955, 1999.

[5] Dennis A Benson, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and David L Wheeler. Genbank. *Nucleic Acids Research*, 36(suppl 1):D25–D30, 2008.

[6] Sergey Brin and Lawrence Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Computer networks*, 56(18):3825–3833, 2012.

[7] Jack Clark. Google Turning Its Lucrative Web Search Over to AI Machines. *Bloomberg Business*, 2015 (accessed 2016-01-21). http://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines.

[8] Alan Kaylor Cline and Inderjit S Dhillon. Computation of the Singular Value Decomposition. *Handbook of Linear Algebra*, pages 45–1, 2006.

[9] Stephen Cook. The P Versus NP Problem. In *Clay Mathematical Institute; The Millennium Prize Problem*. Citeseer, 2000.

[10] Alexandru Dan Corlan. *Medline Trend: Automated Yearly Statistics of PubMed Results for any Query*, 2004 (accessed 2015-10-13). http://dan.corlan.net/medline-trend.html.

[11] Michelino De Laurentiis, Sabino De Placido, Angelo R Bianco, Gary M Clark, and Peter M Ravdin. A Prognostic Model that makes Quantitative Estimates of Probability of Relapse for Breast Cancer Patients. *Clinical Cancer Research*, 5(12):4133–4139, 1999.

[12] Tom Dietterich. Overfitting and Undercomputing in Machine Learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[13] Domo. *Data Never Sleeps Infographic*, 2015 (accessed 2016-02-5). https://www.domo.com/learn/infographic-data-never-sleeps.

[14] Marco Dorigo and LM Gambardella. Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem. In *Proceedings of ML-95, Twelfth International Conference on Machine Learning*, pages 252–260, 2014.

[15] Carl Eckart and Gale Young. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, 1(3):211–218, 1936.

[16] Nina Galanter, Dennis Silva Jr, Jonathan T Rowell, and Jan Rychtář. The Territorial Raider Game and Graph Derangements. *arXiv preprint arXiv:1507.06286*, 2015.

[17] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[18] Gene H Golub and Charles F Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012.

[19] GroupLens. *MovieLens*, 2015 (accessed 2015-11-22). `http://www.grouplens.org/datasets/movielens/`.

[20] F. Maxwell Harper and Joseph A. Konstan. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[21] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1–9. ACM, 2014.

[22] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[23] Ron Kohavi and Foster Provost. Glossary of Terms. *Machine Learning*, 30(2-3):271–274, 1998.

[24] Yehuda Koren. The Bellkor Solution to the Netflix Grand Prize. *Netflix prize documentation*, 81, 2009.

[25] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised Machine Learning: A Review of Classification Techniques, 2007.

[26] Pedro Larrañaga, Borja Calvo, Roberto Santana, Concha Bielza, Josu Galdiano, Iñaki Inza, José A Lozano, Rubén Armañanzas, Guzmán Santafé, Aritz Pérez, et al. Machine learning in Bioinformatics. *Briefings in Bioinformatics*, 7(1):86–112, 2006.

[27] Roderick JA Little and Donald B Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, 2014.

[28] John F Mccarth, Kenneth A Marx, Patrick E Hoffman, Alexander G Gee, Philip O'Neil, M L Ujwal, and John Hotchkiss. Applications of Machine Learning and High-Dimensional Visualization in Cancer Detection, Diagnosis, and Management. *Annals of the New York Academy of Sciences*, 1020(1):239–262, 2004.

[29] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad Click Prediction: A View from the Trenches. In *Proceedings of the 19th Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining*, pages 1222–1230. ACM, 2013.

[30] Michael Minelli, Michele Chambers, and Ambiga Dhiraj. *Big data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses*. John Wiley & Sons, 2012.

[31] Tom M Mitchell et al. Machine learning. WCB, 1997.

[32] David E Moriarty, Alan C Schultz, and John J Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

[33] Markus Müller and Christian Pellegrini. Machine Learning Approaches to Lung Cancer Prediction from Mass Spectra. *Proteomics*, 3:1716–1719, 2003.

[34] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[35] Matthew Richardson, Amit Prakash, and Eric Brill. Beyond PageRank: Machine Learning for Static Ranking. In *Proceedings of the 15th International Conference on World Wide Web*, pages 707–715. ACM, 2006.

[36] C. J. Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[37] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, pages 285–295. ACM, 2001.

[38] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[39] Guy Shani and Asela Gunawardana. Evaluating Recommendation Systems. In *Recommender Systems Handbook*, pages 257–297. Springer, 2011.

[40] R John Simes. Treatment Selection for Cancer patients: Application of Statistical Decision Theory to the Treatment of Advanced Ovarian Cancer. *Journal of Chronic Diseases*, 38(2):171–186, 1985.

[41] Phil Simon. *Too Big to Ignore: The Business Case for Big Data*, volume 72. John Wiley & Sons, 2013.

[42] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*, volume 1. Massachusetts Institute of Technology press Cambridge, 1998.

[43] Theja Tulabandhula, Cynthia Rudin, and Patrick Jaillet. The Machine Learning and Traveling Repairman Problem. In *Algorithmic Decision Theory*, pages 262–276. Springer, 2011.

[44] Joannes Vermorel and Mehryar Mohri. Multi-armed Bandit Algorithms and Empirical Evaluation. In *Machine Learning: ECML 2005*, pages 437–448. Springer, 2005.

[45] Manolis G Vozalis and Konstantinos G Margaritis. Applying SVD on Generalized Item-based Filtering. *IJCSA*, 3(3):27–51, 2006.

[46] Joost Wit. Evaluating Recommender Systems: An Evaluation Framework to Predict User Satisfaction for Recommender Systems in an Electronic Programme Guide Context. 2008.

[47] WordStream. *How Does the AdWords Auction Work?*, 2011 (accessed 2015-10-13). http://www.wordstream.com/images/what-is-google-adwords.pdf.

[48] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving Recommendation Lists Through Topic Diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005.