Context-Aware Rank-Oriented Recommender Systems

by

Brian Ackerman

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2012 by the
Graduate Supervisory Committee:

Yi Chen, Chair
Kasim Candan
Huan Liu

ARIZONA STATE UNIVERSITY

December 2012

ABSTRACT

Recommender systems are a type of information filtering system that suggests items that may be of interest to a user. Most information retrieval systems have an overwhelmingly large number of entries. Most users would experience information overload if they were forced to explore the full set of results. The goal of recommender systems is to overcome this limitation by predicting how users will value certain items and returning the items that should be of the highest interest to the user. Most recommender systems collect explicit user feedback, such as a rating, and attempt to optimize their model to this rating value. However, there is potential for a system to collect implicit user feedback, such as user purchases and clicks, to learn user preferences. Additionally with implicit user feedback, it is possible for the system to remember the context of user feedback in terms of which other items a user was considering when making their decisions. When considering implicit user feedback, only a subset of all evaluation techniques can be used. Currently, sufficient evaluation techniques for evaluating implicit user feedback do not exist.

In this thesis, I introduce a new model for recommendation that borrows the idea of opportunity cost from economics. There are two variations of the model, one considering context and one that does not. Additionally, I propose a new evaluation measure that works specifically for the case of implicit user feedback.

DEDICATION

To my family

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this work.

I would like to thank my thesis adviser, Dr. Yi Chen, for taking me in as a undergraduate student and allowing me to continue with her into my graduate studies.

I would also like to thank my committee members, Dr. K. Selcuk Candan and Dr. Huan Liu. They both offered their time and energy to improve the quality of my work.

I would like to thank all of my labmates over the past two years: Tim, Brian, Ziyang, Siva, Lishan, Yunzhong, “Sean” (Shan), Doug, “Kevin” (Chia-wen), James, and Jadiel. They were always willing to discuss problems and share ideas.

Finally, I would like to thank my high school Computer Science teacher, Mrs. Renee Ciezki. She was always there to provide unique challenges that fueled my passion for Computer Science. She also taught me the lifelong lesson that “weeks of programming can save you hours of planning.”

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

1.1 Motivation

Recommender systems are a type of information filtering system that provides users suggestions on new and exciting items that may be of interest. Recommender systems differentiate themselves from traditional information filtering systems as their suggestions are personalized to each user. These suggestions are directly related to the user decision making process of a particular system. Examples of these decision making processes are choosing which songs to listen to or which items to purchase. Different recommender systems focus on different "*items*". Items can be movies, songs, books, events, people, or news stories. Since each system focuses on different types of items, they each require different techniques of recommendation and their own user interface to make effective and useful suggestions to personalize the user's experience with the system.

Recommender systems operate on a principle commonly seen in everyday life: individuals rely on friends and family to make daily or routine decisions [45, 49]. For example, it is common to ask other people for suggestions on which book to read next or where to travel for a vacation. Because of this, recommender systems seek to mimic this behavior.

Recommender systems are geared towards individuals who may not be a domain expert for the items used by the system. Because of their lack of domain knowledge, it would be difficult for the user to evaluate an overwhelming number of alternatives to choose from. This is often referred to the problem of information overload. For example, Pandora is a music recommender system focusing on discovering new music that contains over

800,000 different songs from a large spectrum of genres [68]. A user may be familiar with only a small subset of this large catalog of songs, but would like to use the system to discover new music. Pandora allows users to select only one musical group or song they like and can instantly begin suggesting songs that the user may also like.

Traditional information filtering systems can also accomplish problems with information overload [10] and lack of domain knowledge [17]. However, these systems do not take into account that different users have varying needs and require special treatment. The diversity of users clearly shows the need for recommender systems. For example, a user may come to a movie recommender system knowing they want to watch a *Johnny Depp* movie. Depp has acted is almost 60 different movies in a diverse set of genres [28]. A traditional information filtering system may return a list of all the movies Depp has acted in. This would solve the problem of information overload as the system could select a small subset of all movies ever made. However, 60 movies may still be overwhelming to the user. In this case, a recommender system could choose an even smaller subset of movies or rank the entire list of movies. This would be done on a personal user basis. Some users may prefer his children-friendly movies while others may like his work with director Tim Burton. Being able to personalize the user's experience cannot be accomplished with traditional information filtering system, but only with a recommender system.

Recommender systems collect user feedback, such as ratings on a scale of 1 to 5 and "*thumbs up*" or "*thumbs down*" data. Based on this user feedback, they attempt to generate predictions on how the *active user*, the user currently seeking suggestions, will value the items associated with the system.

These predictions determine which items the system will recommend to the active user. However, there are many different ways to collect user feedback. After a system decides how it would like to collect user feedback, it can then choose from a variety of recommendation techniques.

The general input for a recommender system consists of a set of items and a set of users. Additionally, there is some information linking these two sets that records users' preferences on the items. A commonly used technique to make recommendations is collaborative filtering which utilizes the entire log of user preferences to make recommendations for the active user. These approaches primarily focuses on user feedback in terms of a numerical score, usually on a scale of 1 to 5. Work on this type of feedback was fueled by the offering of the Netflix Prize and their associated dataset [51]. Netflix asked the question of *how to improve the quality of recommendation*? For this, a proxy objective was defined which was to minimize the root mean square error between the recommender system's predicted ratings and the users' actual ratings. However, this is a limited problem setting where only numerical rating values are possible and results are only evaluated on prediction accuracy.

Although many existing recommender system techniques use rating scores as the mechanism for user feedback and give accurate prediction of user's ratings [19, 33, 59], it is important to note that this is not necessary. First, I discuss when prediction accuracy is not necessary. Since a recommender system is trying to aid in the user's decision making process, its job is merely to suggest items to a user. In this case, a system just needs to decide the best alternatives for the user without providing a predicted relevance value. For example, if a user wants to watch a movie on Netflix, they system only needs to suggest items which can be done by ranking items based on predicted relevance values. It does not matter whether the predicted relevance value for the movie is 5 or only 3. If the user wants to watch a movie, they will watch the movie with the highest perceived value. When returning a list of ranked items to the user, the relative order of the items is the most important characteristic of the output. Next, I discuss the limitations of numerical rating scores. When a user rates an item a 4, it is not inherently saying it is twice as relevant as an item rated a 2. Rating scores are a mechanism to determine the relative order of user's preferences. Working on user feedback on an ordinal scale (e.g. A, B, C, D, F) is an important problem because some recommender systems take feedback in this form and numerical rating values can even be interpreted this way. This makes ordinal relevance feedback a more general problem that can work for a wider variety of systems such as [35, 39, 60, 57].

Even though ordinal feedback is more general than a numeric score, *pairwise preferences* can work on even more types of user feedback. Pairwise preferences are comparisons between items that give their relative order of preference. For example, a user may prefer item i over item j, or it may prefer item j to item j. This feedback can be derived from implicit user feedback such as items a user watches are preferred to those items that a user clicks on, but does not watch. Implicit feedback can be easily obtained through all user interactions without requiring users to do more work such as thinking about a rating score to give an item. This type of feedback is also more common and not as sparse as explicit user feedback which makes its use beneficial. It is also worth noting that pairwise preferences do not restrict the scale in which users can give feedback (e.g. only from 1 to 5, or only A to F). To the best of my knowledge, there is no existing work that handles pairwise preferences.

There is also a benefit from leveraging information regarding the context in which feedback information is generated. Beyond the traditional ideas of context (e.g. weather, time, or location), there is the opportunity to use information in terms of which alternatives a user sees before interacting with the system or making their decisions. This is the context of user choice which relates the decision making process in sessions. Users are given a set of items that they are offered by the system. A user can only act upon a limited number of these items (e.g. purchase, watch) due to certain constraints. Therefore they must give up the ability to act on certain items which is related to the idea of opportunity cost in economics. Opportunity cost is the cost associated with passing up the next best choice when making a decision [22]. Therefore, I can use this information to better learn user preferences when knowing which items a user had to give up when making their decision.

1.2 Contributions

I study a new problem setting for recommender systems to handle pairwise preferences rather than numerical or ordinal user relevance feedback. Pairwise preferences are more general than other types of feedback and can actually be generated based on other feedback (e.g. items rated 5 are preferred to items rated 4). To handle this new problem setting, I propose a new model for item relevance which generates a relevance score to predict whether a certain item is preferred to another item to predict pairwise preferences. This model borrows the concept of opportunity cost from economics. This model can be used to rank a new set of items that a user wants suggestions.

Additionally, I study another problem setting where user feedback is collected within user sessions. I again borrow the concept of opportunity cost from economics. This time I further extend my previous model to handle this new problem setting. The new model looks at incorporating the context of the items shown to the user. This is related to opportunity cost as the user can only act upon as subset of these items and must give up the others. The new model can can generate context-dependent relevance values for items that are based on the other items that are involved within a particular user session. Again, using this model, items can be ranked during a new user session.

I also look at improving evaluation techniques to measure the rank accuracy of a recommender system. There are currently no existing evaluation techniques to handle rank accuracy evaluation for pairwise preferences that are weighted based on the relevance of the item. I propose Expected Discounted Rank Correlation to compute the rank accuracy of partially ordered lists where mistakes in ranking the most preferred items are more heavily weighted than lesser preferred items.

Finally, I run a set of experiments to show the efficacy of my techniques. I run experiments on the global, or pairwise preference, problem setting against existing work to show my methods produce better ranked outputs. I then run more experiments for the session problem setting against the only work that handles sessions. This again shows that my approaches provide a better ranked output.

Chapter 2

RELATED WORK AND BACKGROUND

2.1 Introduction

Recommender systems are a tool for users that provide suggestions on new and exciting items that may be of interest to the user. These suggestions are directly related to the user decision making process of a particular system. For example, some systems may make recommendations on which music to listen to while other may suggest goods to purchase. There are many different ways to make these recommendations to the user. The system attempts to either assign a relevance value to each item or makes pairwise comparisons of items to find those items that should be suggested to the user.

The general techniques used by recommender systems are content-based filtering, collaborative filtering, demographic filtering, knowledge-based filtering, community-based filtering, or a hybrid approach [63]. Content-based approaches look at items that the user has previously liked and then attempts to find similar items to recommend [27, 37, 41, 58, 66]. Collaborative filtering uses feedback from all users to make recommendations for the active user [12, 19, 24, 34, 59, 61, 65, 73]. This often involves trying to find similar users and items. Demographic filtering tries to localize recommendations for the particular profile of the active user [44]. Knowledge-based filtering use domain knowledge to match the users needs to the possible items that can be suggested by the system [13, 42, 62]. Community-based filtering is mainly used in social network applications where the system recommends users to follow or content to consume [43, 70]. These recommendations are based on the social graph of the active user. Finally, there are many applications where these methods can be combined in a hybrid approach [36, 46, 47, 67, 72].

2.2 Relevance Classification

A key characteristic of a recommender system is how it makes predictions for relevance to the user. Since the predictions are based on the input data, predicted relevance usually directly relates to this input data. Recommender systems attempt to predict a relevance value for each user/item pair. For this, user feedback can be collected in a variety of different ways as outlined in the below sections.

User Feedback

The two main ways that systems collect user feedback are either explicit and implicit.

Explicit Feedback

Explicit feedback occurs when the system has a mechanism (e.g. button or combo box) that directly allows the users to give feedback. The most common example of this is with movie rating websites where users can click on a button that says how well they like the movie on a scale of 1 to 5. In these cases, it is reasonable for the system to record a specific relevance value as it was given by the user.

Implicit Feedback

Implicit feedback occurs when the system does not offer a mechanism for the user to give feedback. Examples of implicit user feedback are users clicking on items or records of what videos a user watched. In these cases, the system cannot record a specific relevance value for user/item pairs as the user never provides this information.

Relevance Scales

Beyond explicit and implicit user feedback, different systems use different scales for relevance. These scales are outlined below.

Cardinal Relevance

Cardinal relevance is the most commonly studied form of user feedback in recommender systems. Using cardinal relevance means that when a user rates an item a particular value, the relevance assigned to this item has a distinct numerical value and the difference between these two values is significant. This would mean rating an item a 4 means it is twice as good as a 2. Please note that although it is not necessary, most work on cardinal relevance limits the range of values a using can rate items to be fixed which is a limitation of the datasets they are working on. This is because most systems only allow users to rate items on a fixed scale so that all users confine themselves to the same range of rating values. These types of scales are typical on movie rating websites where users rate on a scale from 1 to 5. Research that uses cardinal relevance feedback typically measures the effectiveness of the system using measures of prediction accuracy such as root mean square error. Examples or work using this type of feedback are [19, 33, 34, 59].

Ordinal Relevance

Ordinal relevances looks at feedback that is on any type of ordinal scale (e.g. A, B, C, D, F). Although each value may not have a numerical value associated with it, the relative position of each discrete value is known (e.g. A is better than B, B is better than C, etc...). It is important to note that this scale does not have to be fixed. However, in most literature a fixed number of ordinal relevance values, such as the letter grading scale, is used. Again, this is ensure that all users are using the same range of rating values. Furthermore, it is important to note that ordinal relevance is not the same as multi-class, nominal classification (e.g. red, green, blue). This is because for a multi-class classification, there does not have to be an order among the possible classes. Examples or work using this type of feedback are [35, 39, 57, 60].

Binary Relevance

Binary relevance presents a unique challenge for classification as it actually depends on how the data is handle by the system. Binary relevance could be cardinal if the system uses cardinal relevance values such as 0 and 1. It could be also ordinal if the system uses two ordinal values such as interacted and not interacted. However, using interacted and not interacted can also be handled as a binary classification problem which is not the same as the relative order of these two classes is not considered. Examples or work using this type of feedback are [50, 60, 75].

Unary Relevance

Unary relevance is similar to binary relevance. The difference is that instead of knowing the items with user interaction and without interaction, only the items that were interacted by the user are known.

Pairwise Relevance

Pairwise relevance is very different from cardinal or ordinal. Pairwise relevance assumes that only pairwise comparisons of items (e.g. item i is preferred to item j) are known. This is most commonly associated with implicit user feedback where it would be difficult, if not impossible to

Table 2.1: Examples of Explicit User Feedback based on Relevance Scales

Scale	**Explicit Feedback**
Unary	User clicks a *Like* button
Binary	User can click either a *Like* or *Dislike* button
Cardinal	User rates items on 1 to 5 scale
Ordinal	User rates items on A to F scale
Pairwise	Ask user to choose between two items

generate a total order of preference among items. The system can use its own ordinal-like scale (e.g. bought, clicked, no interaction). From this, pairwise preferences among items can be generated. For example, all items that are bought are preferred to those that are only clicked. Additionally, items bought or clicked are preferred to those items without any interaction. There is no existing work that directly handles this type of feedback.

Now I look at examples of how these different relevance scales can be applied by the system to collect user feedback. First, I look at Table 2.1 which shows examples for explicit user feedback. Notice that the relevance scales go from unary to pairwise. This is from most restrictive to least restrictive. Unary is very limiting since it only tracks items that are liked by the user. Pairwise preferences are the least restrictive because there is not set cardinal or ordinal scale in which items must be ranked. Because of this, there could be endless levels of preference beyond 1 to 5 or A to F. Pairwise can also be used on binary, cardinal, and ordinal feedback. For example, binary feedback can generate pairwise preference between all *liked* items and all *disliked* items. Pairwise can be generated from cardinal and ordinal by creating preferences between all items rated 5, or A, and all items rated 4, or B.

Table 2.2: Examples of Implicit User Feedback based on Relevance Scales

Scale	**Implicit Feedback**
Unary	User buys an item
Binary	User buys an item after being shown a set of possible alternatives
Cardinal	N/A (Not possible to use cardinal value without explicit feedback)
Ordinal	Use implicit ordinal scale (e.g. bought, clicked, no interaction)
Pairwise	Enumerate all possible combinations from ordinal scale

Now I look at Table 2.2 which shows examples for implicit user feedback. Again, I use the same order showing the most restrictive to least restrictive. Please note that cardinal scales for implicit feedback cannot directly be done because it requires an explicit value which cannot be done by the system because it would only be a guess or estimate. Again, pairwise preferences can be generated from binary, cardinal, and ordinal relevance. However, this cannot be done with unary because only one relevance value is known so preferences cannot be generated. Note that ordinal relevance usually is handled on a fixed scale (e.g. grading A to F) while having pairwise preference does not have this restriction.

Context and Relevance

Another facet of relevance is whether the relevance value assigned to an item is static and does change or is dynamic and can change with the context of the recommendation. Below I discuss the differences between context-independent and context-dependent relevance.

Context-independent Relevance

Context-independent is when relevance information is collected without any temporal data associated with it. Even if the data is collected within a user session, no other information is stored. This is the general case of a recommender system. This means that each user/item pair has a static value that will not change. This means that if a user rates an item, this value is assumed to be the proper relevance value across all possible contexts.

Context-dependent Relevance

Context-aware recommendation studies the problem of having a user's relevance value for an item change depending on the context of the user. Context information collected during the time of user interaction (e.g. time, weather, or location) can be used to change the relevance of an item. A key characteristic of this type of relevance is that user/item pairs can have dynamic values which change with the context.

2.3 Learning to Rank

Learning to rank is a machine learning technique that attempts to learn a model to rank items based on some set of training data. In the context of this work, learning to rank becomes and important part in using the prediction model that assigns relevance values to items. Generally, there are two main alternatives to learn to rank: pairwise raking and pointwise ranking [40].

Pairwise

Pairwise ranking involves assigning a value to each possible pairwise comparison of items. Based on these pairwise comparisons, then you can construct a ranking. For example, if there are two items, i and j. There might be a value for i is preferred to j and j is preferred to i. If one is bigger than the other, then that is the correct way to ranking those two items. Since pairwise ranking looks at making comparisons on an item to item basis, it has a time complexity of $O(n^2)$. This work is typically done when considering binary preferences and learning is done by comparing items in the more preferred class with those in the less preferred class such as the work in [60].

Pointwise

Pointwise ranking involves giving each item a relevance value. The prediction model being learned is optimized to training data by looking at one training instance at a time. Then a set of items can be solely ranked based on their relevance values. The time complexity of pointwise is only $O(n)$ as it only looks at each item once without making any comparison. This is most common when iterating over the set of user ratings as each point is a given rating. Examples of this work are [19, 33, 34, 59].

2.4 Collaborative Filtering

Collaborative filtering is one technique used in recommender systems. It uses historical knowledge of how users value items and based on all of these previous interactions makes recommendations for the active user. The traditional problem setting for collaborative filtering includes the system knowing ratings for a set of user on a set of items. Using this set of ratings, the system attempts to make predictions for user/item pairs that are previously unknown. Using the problem setting, it is common for the set of ratings to be stored in a matrix where rows and columns represent items and users. There are two main varieties of collaborative filtering techniques: memory-based and model-based.

Memory-based methods focus on finding similar users [25, 61, 69] or similar items [16, 38, 65] to the active user or active item, those involved in the current prediction. There is also work done by combining both of these similarity techniques [73]. After finding the similar users or items to use, predictions are made by using the subset of ratings from the similar users or items. Limitations of memory-based techniques are that most of the work needs to be done online because the it is not known until the time of recommendation which items need to be predicted. It is possible to pre-compute all user to user similarities, or all item to item similarities, but this is not always feasible if the number of users or items is too large (e.g. over 1 million).

Model-based methods try to address the limitation of memory-based techniques. Model-based techniques over this because after learning some model, predictions can be made in constant time online when the user wants a recommendation. This is done by developing a model and then employing some machine learning techniques to find the parameters of the model. Examples of model-based methods are [19, 33, 35, 59, 75].

2.5 Regularized SVD

Regularized singular value decomposition (RSVD) was first proposed by Simon Funk in a blog post regarding the Netflix prize [19]. Previous work had look at performing singular value decomposition (SVD) on the rating matrix, but this came with many issues especially relating to rating sparsity. SVD techniques reduce the dimensionality of the original rating matrix. This matrix could involve 17,000 movies and 500,000 users, as seen in the Netflix Prize dataset [19]. This would lead to 8.5 billion entries in the matrix with only 100 million of these entries being filled in with ratings. Sparse matrices do not perform well with SVD techniques. RSVD does not directly perform SVD, but attempts to construct similar matrices of lower dimensionality.

Funk proposed thinking about each user and item as a feature vector. Each of these vectors are of a given length, k. Each feature in an item vector represented how much the item possessed that particular feature. Each user feature represented how important that feature is to that user. Taking the dot product of a user and an item vector would yield a single scalar value which predicts that user's relevance score on the item. Since the number of possible features for an item is endless, latent features are used which do not correspond to any particular real world feature. A user vector can be denoted as ϕ_u where $\phi_u = \langle f_1, f_2, \cdots, f_k \rangle$ and an item vector can be denoted as ϕ_i where $\phi_i = \langle f_1, f_2, \cdots, f_k \rangle$.

The RSVD model, $\phi_u \cdot \phi_i$, is learned based on an optimization goal and can predict ratings on non-rated items based on this model. To find the feature values, the optimization goal in Equation 2.1 can be used. The difference between the actual user rating, r_{ui}, for user u on item i, is compared with the predicted value, $\phi_u \cdot \phi_i$. This is done for all known ratings, $\mathcal{R}$. Regularization is controlled based on the parameter ρ. Regularization is done to reduce the complexity of the model which prevents over-fitting of the model. To do this, the magnitude of each feature vector is minimized for each user in the set of users, $\mathcal{U}$, and each item in the set of items, $\mathcal{I}$.

$$\min_{\phi^*} \sum_{r_{ui} \in \mathcal{R}} \|r_{ui} - \phi_u \cdot \phi_i\|^2 + \rho \left[\sum_{u \in \mathcal{U}} \|\phi_u\|^2 + \sum_{i \in \mathcal{I}} \|\phi_i\|^2 \right] \tag{2.1}$$

2.6 Work on Ordinal Feedback

There has been some work done with user feedback that is ordinal in nature. The key difference of this work is that relevance feedback is only treating as ordinal, having a relative order, rather than having a numerical value. Work done in this area includes OrdRec [35], EigenRank [39], Bayesian Personalized Ranking(BPR) [60], and Ordinal Matrix Factorization(OMF) [57]. I choose to to further describe OrdRec as it is the latest proposed method and it is extensible to allow for cardinal feedback to be interpreted as ordinal.

Additionally, BPR only handles binary preferences, OMF is overly complex, and EigenRank uses out-dated memory-based techniques.

OrdRec

Koren and Sill [35] proposed a collaborative filtering framework called OrdRec which can be used as a wrapper to any traditional matrix factorization techniques such as RSVD [19] or SVD++ [33] to treat user feedback as as ordinal rather than the traditional explicit relevance score. For example, all items giving an a rating of 5 are preferred to those given a rating of 4 or lower. The focus of their methods is to provide a personalized item rating distribution for each user. They say that this provides a richer system output as it can predict the mean, mode, or median for any rating value. Additionally, it can be used by the system to provide a confidence level for its predictions.

The input data for OrdRec is similar to that of RSVD, presented in Section 2.5, as their are users, items, and ratings from users on items. The difference is that instead of taking in cardinal rating values, OrdRec takes in ordinal values. They refer to the possible rating values as 1, 2, 3, ..., $\mathcal{S}$.

Since OrdRec is not a model, but a framework, it requires a recommendation model whose internal rating score is denoted as y_{ui} for a predicted rating for user u on item i. They have $\mathcal{S} - 1$ thresholds corresponding to each of the possible rating values, except for the most relevant, $\mathcal{S}$. These thresholds are denoted as $t_1, t_2, ..., t_{\mathcal{S}-1}$ such that $t_1 \leq t_2 \leq ... \leq t_{\mathcal{S}-1}$. The first threshold value is a parameters of their framework with the others being set using a set of parameters, $\beta_1, \beta_2, ..., \beta_{\mathcal{S}-2}$. These values set the thresholds such that $t_{r+1} = t_r + \exp(\beta_r)$. They denote the combined parameters of their framework, t_1 and $\beta_1, \beta_2, ..., \beta_{\mathcal{S}-2}$, and those from whichever model being used as Θ.

For their method, they first generate a random score, z_{ui} that is generated based on a normal distribution centers at the internal score of the modeling being used, y_{ui}. They assume that this random score corresponds to the ordinal value which it falls according to the threshold value such that $t_{r-1} < z_{ui} \leq t_r$. This can be used to say that the probability of the user rating an item as r is $P(r_{ui} = r|\Theta)$ which is equal to $P(t_{r-1} < z_{ui} \leq t_r)$. Using this, they define the probability of observing a rating $r_{ui} = r$ as follows:

$$P(r_{ui} = r|\Theta) = P(r_{ui} \leq r|\Theta) - P(r_{ui} \leq r-1|\Theta) \tag{2.2}$$

Their goal is to learn the function $L(\mathcal{R})$ which is the log likelihood of $r_{ui} = r$ using the training set of ratings, $\mathcal{R}$. They update all parameters of this function, Θ, using stochastic gradient descent. This can then be used to determine a probability distribution over the possible rating values.

After having the probability distribution, items still need to be ranked for the user. This is done by creating a vector for each item/item pair which is of the length $\mathcal{S}$. Each index is the difference in probability between the two items for each rating value. This can then be used to learn a linear mapping to give a predicting rating for each item.

2.7 Context-Aware Recommendation

Context-aware recommendations are those that use *contextual information* (e.g. time, location, weather) when making predictions [5]. This information can be used to generate context-dependent relevance values for items as discussed in Section 2.2. In general, there are three main alternatives to incorporating context with a model for a recommender system. These alternatives are pre-filtering, post-filtering, and contextual modeling [5]. These alternatives are described below.

Pre-filtering

Pre-filtering consists of taking the traditional input data and only selecting a subset of this data before learning a model or making predictions. This subset of the input data is chosen based on some selection criteria which compares the current context to previous contexts to find the most similar historical contexts. For example, in [3], both time and location are considered as possible third dimensions of the rating matrix. Now there are rows and columns for users and items and entire matrices for different days of the week or cities. If I consider the matrices to be for days of the week, the current day may be Sunday, so pre-filtering would only look at the rating matrix for all ratings made on Sundays. If a model is used, then it is learned based only on the subset of data which will result in context-dependent relevance values. However, since the selection of a subset of data is dependent on the current context, the entire model or prediction process has to be done online. This would cause this method to be extremely costly with many online calculations and the delay would be easily noticeable by users. Pre-filtering is seen in the following works: [3, 4, 30].

Post-filtering

Post-filtering consists of the first learning a model or making predictions in the usual manner. After this model is learned or initial predictions are made, which can be done entirely offline, refinements can be made in a variety of ways. Looking at the example from pre-filtering that uses multi-dimensional matrices, I could learn the model and then update the model further by only learning the model or doing calculations based on the similar contexts (e.g. same day’s ratings). This would change the model temporarily for the active user or edit the predictions, again resulting in context-dependent relevance values. The main advantage of this approach is that it may be practical as most of the work is done offline and only small refinements are required during online computation. Nonetheless, to make substantial changes would improve the quality, but this would still take longer than a user would be willing to wait. Post-filtering is used in the following works: [4, 55, 52, 53].

Contextual Modeling

Context modeling has a range of alternatives. All would not do any pre-filtering or post-filtering. This approach requires a model that would take the context as a parameter. The model could be learned offline and when predictions are needed for the active user, no change would be made to the model and the prediction could be made immediately. This seems to be the most viable alternative as predictions can be made in constant time with no notable delay to the users. Contextual modeling is used in [3, 8, 56, 65].

Contextual modeling is the most applicable for my work as I was to create a contextual model where the context is the set of items shown to a user as the time of recommendation. Because of this, I give more information regarding how context is handle in this type of work.

Contextual modeling looks at incorporating contextual attributes such as day, time, or location directly into the model. One approach for this is used in [56]. This approach creates a context profile that links user's to contexts. This context profile is used to model how a user interacts with the system in different context (e.g. different days of the week). It captures a user's tendency to interact with particular items and how they are valued depending on the depending on the context. The profile will map a user and a context with the relevance values for the set of items observed in that context. Since this paper is a framework that can be used with different models, they do not describe any specific learning techniques linked to this type of user profiling, but rather talk about how it can be used in collaborative filtering to restrict learning the model for a user to only take into consideration similar user and similar contexts.

2.8 Collaborative Competitive Filtering

Yang et al. [75] proposed collaborative competitive filtering (CCF) as a way to predict binary user actions based on historical user sessions. This is different from other work as it looks at exploiting the context of the items displayed to the user. This changes the problem setting from previous work to have a set of historical user session instead of ratings. Each session consists of an offer set of items, $\mathcal{O}$, and a decision set of items $\mathcal{D}$. The offer set are the items displayed to the user while the decision set is the items they user acted upon (e.g. purchased). Note that this is only binary relevance and CCF imposes the limitation of only having one item in the decision set. That is that only one item can be considered to the most relevant in any given offer set.

CCF uses the idea of opportunity cost, borrowed from economic theory. Their assumption is that each item has a potential revenue to the user. This revenue is gained by taking the one of the opportunities and passed up otherwise. They say that each user will try maximize their profit by choosing the item that has the most revenue in comparison to potential loss that might occur by not taking an opportunity. They use this assumption to attempt to assign a higher relevance to items known to be in the decision set in comparison to the items that are not in the decision set. They do this by saying that the revenue from the chosen item is higher than the maximum of the non-choosen items such that $r_{ui^*} \geq \max\{r_{ui} | i \in \mathcal{O} \setminus \mathcal{D}\}$ where r_{ui} is the revenue of an item and r_{ui^*} is the revenue of the item in the decision set. The revenue in their model is modeled using the same latent features vectors as seen in Section 2.5.

As CCF does not have their own model, they update the parameters of the traditional latent feature vector based on certain update rules. Like the RSVD and OrdRed, they use stochastic gradient descent for to update the parameters. CCF updates these parameters based on the following optimization goal:

$$\min \sum_t \log \left[\sum_{i \in \mathcal{O}_t} \exp(\phi_{ut} \cdot \phi_i) \right] - \phi_{ut} \cdot \phi_{it}^* \qquad (2.3)$$

This goal looks at all sessions at different times denoted as t. This method uses the assumptions of a multinomial logit model.

2.9 Evaluation of Recommender Systems

Evaluation is important in any field with recommender systems not being an exception. Recommender systems evaluation is generally broken up into two different categories: user-centric or system-centric. User-centric evaluation of recommender systems is done with respect to how a user would judge the quality of the system. There are objective ways of doing this such as prediction and ranking accuracy. There are also subjective measures of doing this such as novelty, diversity, serendipity, and trust. System-centric evaluation determines the quality of the system from the system's perspective. Examples of system-centric measures are coverage, robustness, confidence, utility, risk, privacy, adaptability, and scalability.

My work specifically look at accuracy with an emphasis on ranking. Most of the existing work looks at improving the prediction accuracy which means optimizing to an explicit relevance value. However, there are many times when only implicit feedback is collected and in these cases, existing measures and metrics fail to work. In these cases, rank accuracy measures and metrics can be used. However, some of them require information that might not also be provided using implicit feedback such as nDCG [29] needing explicit relevance values.

The next sections review common characteristics of measures and metrics and some commonly used metrics and measures used for rank accuracy.

2.10 Characteristics of Measures and Metrics

Although all measures and metrics attempt to show the efficacy of a particular system or method, they each have different characteristics. In terms of rank accuracy, all measures and metrics looks at comparing how well the system ranks items with respect to how the user would rate the same set of items. One of these rankings is the prediction while the other is the ground truth. These measures and metrics has characteristics themselves and characteristics in terms of what type of data they can evaluate. First I look at characteristics of the evaluation technique itself.

Measures vs. Metrics

Although the words measure and metric often are used interchangeably, they both have a unique characteristics which make each distinct. The notion of metrics comes from mathematics where they are used as a way to find the distance between two sets. As it is a way to find distance, it is natural to assume that it does not matter which set would be considered the ground truth and which is the prediction. Either way, any metric produces the same result by swapping the ground truth and prediction. This is not the case for a measure. A measure does not guarantee that this holds true and the resulting value is dependent on which ranking is the prediction and which is the ground truth. For example, there may be two rankings, X and Y. There is also a function f that evaluates the differences between these two rankings. In Equation 2.4, it is evident that the order of the parameters of the function do not matter and this constitutes a metric.

$$f(X, Y) = f(Y, X) \tag{2.4}$$

However, in Equation 2.5, it is seen that if the the parameters of the function are ordered one way with a result of some constant c, it is not implied that swapping the parameters will also yield this same constance.

$$f(X, Y) = c \nRightarrow f(Y, X) = c \tag{2.5}$$

Weighted vs. Unweighted

A commonly desired characteristic of a measure or metrics is weighting the most preferred items more heavily. This is because it is most likely that users will only care about the top ranked items. Mistakes at the top ranked items could be costly to the system while making mistakes on very low ranked items may have little to no system impact. Unweighted evaluation techniques will not care about this characteristics while weighted ones will. Weighted measures and metrics will penalize a system heavily on the most preferred items and will penalized a smaller amount for the lesser preferred items.

Besides characteristics regarding the evaluation technique, the are also characteristics regarding what type of data a measure or metric can evaluate.

Total Order v. Partial Order

Another characteristic is the ordering of the ranking data. In the case of a total order, there exists a relative order between each pair of items in the ranked set of items. However, in the case of a partially order list, they may be some pairs of items where no order exists. For example, assume the succeeds operator, $\succ$, denotes that a relative order exists between two items and there is a set of fmy items, A, B, C, and D. In the total ordered list below, there is order between each pair of item. In the partially ordered list, there is not order between the items B and C. This means the relative order of these two items is unknown.

Total Order $A \succ B \succ C \succ D$

Partial Order $A \succ \{B, C\} \succ D$

The ordering for a particular system is a result of the input data given to the system. If the input data has total order information, then it is reasonable for the system to output a total order. On the other hand, if the system has incomplete ordering information in the case of a partial ordering, then the system most likely will only produce a partially ordered ranking as an output.

Known Relevance vs. Unknown Relevance

Relevance is another characteristic of the ranking data. Known relevance is when each item to be ranked has an explicit value associated with it. For example, in the Netflix dataset, each user-movie pair has a numerical rating associated with it on a scale from 1 to 5. However, there are cases where users can give an explicit feedback and there is still unknown relevance. This is the case where users are rating on an ordinal scale such as A, B, C, D, F. Unknown relevance also occurs when the user gives implicit feedback or only pairwise preferences are collected by the system.

2.11 Existing Measures and Metrics

There are currently an overwhelming large number of measures and metrics for evaluating rank accuracy of information retrieval systems. Below is a list of commonly used measure or those proposed to address issues with others. Some work on a variety of problem settings while others only work on very limited problem settings.

Pairwise Loss A very common evaluation technique that often comes with a different name such as Area under the Curve (AUC) [60] or Frequency of Concordant Pairs (FCP) [35]. In all cases, it measures the number of pairwise inversions in a list.

Normalized Discounted Culumlative Gain [29] A weighted measure that works with only explicit relevance feedback on a cardinal scale.

Normalized Distance-Based Performance Measure (NDPM) [76] A non-weighted measure that also requires explicit relevance feedback on a cardinal scale.

Spearman's Rho (ρ) [71] and [Kendall's Tau (τ) [31] Highly correlated measures that can handle ties in ranking orders based on two list of ranked items.

R-Score [12] Weighted measure that handles cases where users are presented with a long list of items, but they will only interact with one item or a very small set of items that are highly ranked on the list. It uses exponential decay to yield a worse results if the user interacts with an item ranked low on the list.

Mean Reciporcal Rank (MRR) [6] Similar measure to Average Reciprocal Hit Rank (ARHR) [16] that look at the index of an item in a ranked list that is chosen by the user. It then uses the reciprocal value (e.g. $\frac{1}{n}$ for index of n) for its evaluation.

AP Correlation [77] Weighted metric that requires total ordered ranked lists and then takes the average precision at every index in the list.

Now I will examine three measures and metrics that can be useful on my particular problem settings.

Pairwise Loss

Pairwise loss is a metric that determines the number of pairwise inversions between two ranked lists. It is also sometimes referred to as the area under the curve or frequency of concordant pairs. Pairwise loss is also an easy way to measure whether a preference is held when considering a set of pairwise preferences and can be used to evaluate only a single preference. The function that can measure the loss of a single preference is given in Equation 2.6. The loss function is denoted be l with the input being a preference p. If the preference is held, then the output is 0. If the preference is not held, or is violated, the output is 1.

$$l(p) = \begin{cases} 0 & : preference\ held \\ 1 & : preference\ violated \end{cases} \tag{2.6}$$

I can calculate loss based on the set of preferences given by a ranked list. For example, I can look at the preference generated in Table 2.3. I can see that the

Table 2.3: Example Preferences

Predictions	Ground Truth
$A \succ B$	$A \succ B$
$A \succ C$	$A \succ C$
$C \succ B$	$B \succ C$

Table 2.4: Example Rating and Preference Data

$\mathcal{I}$	$rel(i)$	$index(i)$	$\widehat{index}(i)$
A	5	1	2
B	4	2	1
C	3	3	3
D	2	4	4

only violated preference is based on the ground truth is $B \succ C$ as the prediction is $C \succ B$. Based on this I know that one third of the preference are violated and the loss for this list is $\frac{1}{3}$.

Normalized Discounted Cumulative Gain

Cumulated gain-based evaluation was proposed by Jarvelin and Kekalainen in 2002 [29] to evaluate rank accuracy when the ground truth is a set of user ratings. The intuition is that it is more important to correctly predict highly relevant items than marginally relevant ones.

nDCG is formally defined in Equation 2.7. I denote $\mathcal{I}$ as a set of items suggested by the system, $rel(i)$ is the relevance of item i according to the user, $index(i)$ is the index of item i sorting the items based on the user's relevance score, and $\widehat{index}(i)$ is the index of item i sorted by the system's prediction.

$$nDCG = \frac{1}{Z} \cdot \left[\sum_{i \in \mathcal{I}} \frac{2^{rel(i)} - 1}{log_2(1 + \widehat{index}(i))} \right] \tag{2.7}$$

Z gives the maximum possible discounted cumulative gain if the items were correctly sorted, and is used as a normalization to ensure the result is between

0 and 1.

$$Z = \sum_{i \in \mathcal{I}} \frac{2^{rel(i)} - 1}{log_2(1 + index(i))} \tag{2.8}$$

For example, I look at the sample data in Table 2.4. If I look at item A, $rel(A)$ = 5, $index(A)$ = 1, and $\widehat{index}(A)$ = 2. Below is the full sample calculation for $nDCG$.

$$\begin{aligned} nDCG &= \tfrac{1}{Z} \cdot \left[\tfrac{2^4-1}{log_2(2)} + \tfrac{2^5-1}{log_2(3)} + \tfrac{2^3-1}{log_2(4)} + \tfrac{2^2-1}{log_2(5)}\right] \\ &\approx \tfrac{1}{Z} \cdot 39.351 \\ Z &= \tfrac{2^5-1}{log_2(2)} + \tfrac{2^4-1}{log_2(3)} + \tfrac{2^3-1}{log_2(4)} + \tfrac{2^2-1}{log_2(5)} \end{aligned} \tag{2.9}$$

This gives a value of .870 for the example.

nDCG assumes that I know an actual relevance for each item. It does not directly support the case when an item is preferred to another item, but the magnitude of the preference is unknown. When there is a total order on item preferences, I may assign rating scores with equal magnitude and then use nDCG. However, the absence of total order will result in such score assignments to be impractical, thus showing inapplicability of nDCG.

AP Correlation

AP (average precision) correlation (τ_{ap}) was proposed by Yilmaz et al. [77] as a modification to Kendall's τ which penalizes mistakes made for highly relevant items more than less relevant items. AP correlation finds the precision between two total orders at each index in the list and then takes the average of these values, as defined in Equation 2.10.

$$\tau_{ap} = \frac{2}{N-1} \cdot \left[\sum_{i \in \mathcal{I}} \frac{C(i)}{index(i) - 1}\right] - 1 \tag{2.10}$$

N is the number of ranked items in the list and $C(i)$ is the number of items at an index less than $index(i)$ that are correctly ranked according to the ground truth. Consider the data in Table 2.4. For item A, $C(A)$ = 0 because A comes after B in the prediction, but A is preferred to B in the ground truth. Below is a sample calculation.

$$\tau_{ap} \;=\; \tfrac{2}{4-1} \cdot \left[\tfrac{0}{1} + \tfrac{2}{2} + \tfrac{3}{3}\right] - 1 \;=\; \tfrac{1}{3} \tag{2.11}$$

AP correlation is measured on scale of -1 to +1, where -1 means the lists are in reverse order and +1 means the list are the same. AP correlation assumes that each list, the ground truth and the system's prediction, gives a total order of items. There is no simple modification of AP correlation to support partial orders.

2.12 Preference Elicitation and CP-nets

Preference elicitation is an important issue in the field of recommender systems. An overview of these methods are found in [15]. Traditional elicitation methods try to give a value proposition to the user to determine which features they most value by asking questions to determine user's preferences. This is very time consuming on the part of the user as it requires then to think about how they value many different features. Another type of elicitation is mentioned where the the system may make recommendations and then give the user options to say the features in the item they most value and how they value them (e.g. larger value or smaller value). This allows the system to learn a user's preferences and present better items. The paper also briefly mentions on the typical elicitation methods such as a user rating. There is also another work, [20], that looks at trying to elicit minimal feedback in order to maximize preference learning. This work assumes that only partial orders are known over a set of items. In order to better learn user's preferences, they try to elicit only small amounts of user feedback to better make predictions for which items a user will value the most. There has also been work done in [14] that allows users to explicitly give negative feedback. Although the paper looks at ontologies, they could easily be considered preference graphs where edges are used to designate a user's preference. The negative feedback used in the paper would be explicit user feedback so it goes beyond the scope of my work. However, there may be special cases of implicit negative feedback (e.g. session with no user interaction) where this work could be applicable as discussed later in future work.

CP-nets are a representation of conditional preferences that can be used to determine how a user may value an in given contexts. They can capture user's preferences based on different decisions that user has previously made. For example, a user needs to make two different decisions: one for which wine to order and the other regarding which entree to order. If the user has previously made the decision to order beef as an entree, this will have an effect on which type of wine they will order. CP-nets can capture this by saying that if a user has order beef, then they will value red wine more than white wine. CP-nets are very useful in the case of sequential decisions where the outcome of one decision has an impact on future decisions. However, in my work, I consider each decision to be independent. Work on CP-nets can be found in [11, 64].

Preference elicitation and CP-nets go beyond the scope of my work, but may be applicable for future work which is discussed in Chapter 7.

Chapter 3

GLOBAL METHODS

3.1 Introduction

As mentioned previously, traditional work with recommender systems looks at using cardinal relevance values, or numerical ratings, to learn a user's preferences. Newer work has relaxed this constraint by working on ordinal relevance values such as a letter grading scale. However, there is a more general problem setting that will work for both of these cases and allow for more expressibility for user's preferences.

Implicit user feedback is easily available in large quantities as user's generate this information each time they interact with the system. Sparsity of explicit user feedback, such as ratings, is a known and active problem in the recommender system community [1, 9, 26]. Implicit user feedback can alleviate the sparsity problem and provide more input data for a system to learn user's preference. Since implicit user feedback does not collect explicit relevance values, it is natural to generate pairwise comparisons of items, or pairwise preference, from this feedback.

As mentioned previously, pairwise preferences can be generated based on implicit user feedback. For example, I can look at the sample feedback generated in Table 3.1. Here I can see that some movies were purchased, others were clicked on, and some have no user interaction. Based on this implicit feedback, I could generate the preferences: *Forrest Gump* is preferred to *Green Mile* and *Catch Me If You Can* is preferred to *Toy Story*, among others.

Pairwise preferences is a more general problem setting than working work ordinal or cardinal relevance feedback. Ordinal and cardinal feedback is restrictive in terms of their scales. Pairwise preferences are more general in

Table 3.1: Sample User Query "*Tom Hanks actor*"

Movie	User Action
Forrest Gump	purchased
Cast Away	purchased
Green Mile	clicked
Catch Me If You Can	clicked
Toy Story	no interaction
Saving Private Ryan	no interaction

the sense that they allow for more possibilities for relevance. For example, if I have 100 items, a cardinal or ordinal scale can only given them a certain number of relevance values (e.g. 1, 2, 3, 4, 5, or A, B, C, D, F). However, pairwise preferences can be constructed such that I know the relative order of all 100 items. This can create a challenge in how to handle the possibility where a system can know a relative order between every item. This is because unlike cardinal or ordinal values, which can easily be stored in a rating matrix, pairwise preferences cannot. A rating matrix consists of rows and columns representing users and items. Each entry is a user's rating on a given item. In order to have pairwise preferences in a rating matrix and use existing techniques, I would have to add another dimension to the matrix. This extra dimension would be for items and the two dimensions for items could be used to record preference. However, this increases the size of the rating matrix by the cardinality of the set of items offered by the systems. This would also be susceptible to sparsity which is not handled well by collaborative filtering. Because of this, additional effort is needed to be able to handle pairwise preferences as input.

This chapter will look at the unique proposed problem setting, the new pairwise opportunity cost model, and its associated learning techniques.

3.2 Problem Definition

The input of this type of system is a set of users, set of items, and users' pairwise preferences regarding the items. The output of the system is a ranking order for the active user interacting the system. This user comes to the system seeking recommendations on a new set of items and the system will return these items in ranked order. The set of users is denoted as $\mathcal{U}$, the set of items is denoted as $\mathcal{I}$, and the set of pairwise preferences is denoted as $\mathcal{P}$.

Definition (Pairwise Preference)**.** *A pairwise preference is a relationship between two items that captures how a user's relative value for the items. For example, if two items, i and j, are considered by a user, there are three possible relationships: i is preferred to j, j is preferred to i, or there may not exist any relationship. A preference between two items is denoted using the succeeds operator ($\succ$). This means that i being preferred to j is denoted as $i \succ j$ and j being preferred to i is denoted as $j \succ i$. It is possible that no relationship exists between two items because the relationship is unknown. For example, in Table 3.1, both Forrest Gump and Cast Away are purchased by the user. There is no way for this system to determine the relationship between these items. Because of this, no preference regarding these two items would existing in the set of pairwise preferences. Note that if explicit feedback is used, it may be possible to say that these two items are equally valued so a relationship is known, but this is not the case for implicit user feedback.*

Preferences can be generated based on implicit user feedback. For example, I can look at the preferences found in Table 3.2 generated from the sample user query "*Tom Hanks actor*." I use *FG* for *Forrest Gump*, *CA* for *Cast Away*, *GM* for *Green Mile*, *CMIYC* for *Catch Me If You Can*, *TS* for *Toy Story*

Table 3.2: Preferences Generated from Sample Query – “*Tom Hanks actor*”

Movie	Less Preferred Movies
FG	GM, CMIYC, TS, SPR
CA	GM, CMIYC, TS, SPR
GM	TS, SPR
CMIYC	TS, SPR
TS	–
SPR	–

and *SPR* for *Saving Private Ryan*. I can see that there are 12 preferences that can be generated from the user actions collected from this session. I can see that the movies that are purchased are preferred to all other movies and their is an unknown relationship among the two movies that were purchased.

The output of such as system would be a ranking order among a set of items. I choose to create a system that minimizes the pairwise loss between the preferences found in the predicted ranking order. I choose pairwise loss because other alternatives, such as nDCG, require explicit relevance values to be known. Pairwise preference are more general where explicit relevance values are not known. Therefore, I use pairwise loss as the objective in my optimization function. It is also a commonly used metric to evaluate such systems as seen in [35, 57].

In order to achieve this, I set an optimization goal for my system, I look at minimizing the pairwise loss over each preference found in the set of preferences. I look at doing this for each user and their individual set of preferences denoted as u and $\mathcal{P}_u$ respectively. This can be achieved by changing the parameters of the chosen learning and prediction model. I denote the set of parameters of any model as Θ. Pairwise loss is measured by the Heaviside loss function seen in Equation 3.2. This function measures the loss for a single preference, $p_{ui|j}$. This preference consists of two items, i and

j, where i is preferred to j. In order for the preference to hold, the predicted relevance value for i being preferred to j must be greater than j being preferred to i. The predicted relevance for i being preferred to j is denoted as $\hat{r}_{ui|j}$ and the predicted relevance for j being preferred to i is denoted as $\hat{r}_{uj|i}$.

$$\min_{\Theta} \sum_{u \in \mathcal{U}} \sum_{p_{ui|j} \in \mathcal{P}_u} l(p_{ui|j}) \tag{3.1}$$

$$l(p_{ui|j}) = \begin{cases} 0 & : \hat{r}_{ui|j} > \hat{r}_{uj|i} \\ 1 & : \hat{r}_{ui|j} \leq \hat{r}_{uj|i} \end{cases} \tag{3.2}$$

Looking back at the example preference found in Table 3.2, I would like to show the set of inequalities that I would like to enforce based on this optimization goal. I simplify the names of the movies to improve readability in the following set of inequalities. I use *FG* for *Forrest Gump*, *CA* for *Cast Away*, *GM* for *Green Mile*, *CMIYC* for *Catch Me If You Can*, *TS* for *Toy Story* and *SPR* for *Saving Private Ryan*. Below is the set of inequalities I would like to hold.

$$\hat{r}_{uFG|GM} > \hat{r}_{uGM|FG} \tag{3.3}$$

$$\hat{r}_{uFG|CMIYM} > \hat{r}_{uCMIYC|FG} \tag{3.4}$$

$$\hat{r}_{uFG|TS} > \hat{r}_{uTS|FG} \tag{3.5}$$

$$\hat{r}_{uFG|SPR} > \hat{r}_{uSPR|FG} \tag{3.6}$$

$$\hat{r}_{uCA|GM} > \hat{r}_{uGM|CA} \tag{3.7}$$

$$\hat{r}_{uCA|CMIYM} > \hat{r}_{uCMIYC|CA} \tag{3.8}$$

$$\hat{r}_{uCA|TS} > \hat{r}_{uTS|CA} \tag{3.9}$$

$$\hat{r}_{uCA|SPR} > \hat{r}_{uSPR|CA} \tag{3.10}$$

$$\hat{r}_{uGM|TS} > \hat{r}_{uTS|GM} \tag{3.11}$$

$$\hat{r}_{uGM|SPR} > \hat{r}_{uTS|SPR} \tag{3.12}$$

$$\hat{r}_{uCMIYC|TS} > \hat{r}_{uTS|CMIYC} \tag{3.13}$$

$$\hat{r}_{uCMIYC|SPR} > \hat{r}_{uTS|CMIYC} \tag{3.14}$$

Notices that there are no inequalities relating the movies *Forrest Gump* and *Cast Away*. This is because there is not a pairwise preference based on the implicit user feedback to suggest this. Only generated pairwise preference will show up in the list of inequalities.

There are two things worth mentioning about this problem setting. First, I do not handle or worry about the possibility of cyclical preferences (e.g. $A \succ B$, $B \succ C$, $C \succ A$). These may be present in the input set of pairwise preferences. However, I do not change my learning methods to use this knowledge. The second item to mention is that I do not prune any pairwise preferences. That is to say if I know the order of three items, A, B, and C, it may generate the preferences $A \succ B$, $A \succ C$, $B \succ C$. I keep the preference $A \succ C$ although it can be implied based on the other two preferences. I do this as it reinforces the preference $A \succ C$ as the relative order of items A and C will be evaluated during my experiments.

3.3 Opportunity Cost Model

Models are used in the learning and prediction process of a recommender system. For the learning process they are learned in accordance with the optimization goal. The model is then used to predict relevance values for the active user at the time of recommendation.

I introduce a new model that can be used to determine which of two items is more preferred. My model attempts to give the value of an item given another item for a particular user. For example, given two items, i and j, my model would assign a relevance value to choose i over j and one for choosing

j over i. The larger of these relevance value predicts which decision the user will make. This model borrows the idea of opportunity cost in economics.

Definition (Opportunity Cost)**.** *Opportunity cost is the cost associated with passing up the next best choice when making a decision [22].*

Opportunity cost is used in many decision making processes. For example, I may be determining what to do during a one hour period on a Saturday afternoon. I may have two options: sit in front of the TV or workout in the park. It is assumed that I can only perform one of this actions during this one hour period. If I choose to sit in front of the TV, I will have to give up being able to workout and vice versa. In choosing one of these opportunities, I will gain some benefit, such as the pleasure of watching TV or health benefits from exercise. However, to gain this benefit, there is an associated opportunity cost which is the benefit from the passed up opportunity. I have to make a decision by weighing which alternative gives me the most benefit while passing up the other activity. If an activity has a large benefit, then passing up that opportunity shows that the chosen activity must have a lot of value to me.

I now introduce the opportunity cost model which borrows its name from the idea of opportunity cost in economics. In the context of this model, the relevance (or value) of an item is based on its own benefit to the user and the opportunity cost of passing up another. It is assumed that users make good economic decisions which are those that show the user preferring items with a higher relevance with respect to other items. When comparing items, a user will prefer the item with more value with respect to other items. Looking at two items, I try to assign a relevance value to each item. For a particular item, its relevance value is its benefit to the user and the opportunity cost of giving up the other item. Looking at Equation 3.15, the relevance value of item i is being assigned based on item j. This is denoted as $\hat{r}_{ui|j}$. The benefit of item i is denoted as b_{ui} and the opportunity cost of item j is denoted as c_{uj}.

$$\hat{r}_{ui|j} = b_{ui} + c_{uj} \tag{3.15}$$

I extend this model to allow for each item and user to have a vector of features. These features will be the traditional *latent features* used in other works such as [19, 59, 75]. The latent features do not correspond to any real world attributes, but rather are used to reduce the number of parameters in the model. To do this, the length of each vector is set to the same length, k. Each index of an item's latent feature vector represents to what extent the item exhibits that particular feature. Each index of a user's latent feature vector represents to what extent the user cares about that feature. In order to find the benefit that an item has to a user, I can take the dot product of these two vectors. Below in Equation 3.16, I show the extension of the opportunity cost model that allows for latent feature vectors. The latent feature vector for user u is denoted as ϕ_u and for item i, ϕ_i. The benefit of an item for a particular user is combination of traditional latent feature vectors for item i and user u where the benefit b_{ui} is equal to $\phi_u \cdot \phi_i$. When the dot product of this two vectors is computed, a single scalar value representing the benefit of this item is output. The opportunity cost of the other item j is also related it its benefit. However, directly taking that value does not make intuitive sense because when giving up an item, the user is not weighing its benefit the same as the item being chosen. This is because the user will not directly receive the benefit from the other item without taking it (e.g. purchasing). Therefore, there is an opportunity cost discount factor denoted as α_u which is applied to the benefit of j. So the opportunity cost of item j is c_{uj} and is equal to $\alpha_u\left(\phi_u \cdot \phi_j\right)$.

$$\hat{r}_{ui|j} = \phi_u \cdot \phi_i + \alpha_u \cdot [\phi_u \cdot \phi_j] \tag{3.16}$$

3.4 Learning the Model

Before the model can be used to make predictions, the parameters must be learned. There are many different ways that this can be done based on various learning to rank methods. A classic learning to rank method is RankSVM [23]. RankSVM is used for ranking search results based on feature vectors for each item (or page) that is being ranked. However, this cannot be used for items as the number of features for items would be difficult to limit to a reasonable amount because of the vast number of actors, directors, genres, and other features. Because of this, latent features are used which are not compatible with RankSVM which needs a fixed feature vector as the input to the system. Additionally, there are other alternatives which require the same set of input feature vectors. However, there are some alternatives such as stochastic gradient descent [32] and stochastic gradient boosting trees [18]. Both of these method would be applicable on my model. Stochastic gradient descent is widely used in recommender systems to learning the parameters of a model as seen in [19, 34, 35, 33, 59, 75]. I choose stochastic gradient descent as boosting method would use weighting which is not necessary or applicable with my particular problem setting. It is also worth mentioning that math and statistic software, such as MatLab [48] and Mathematica [74], have packages to solve these types of problems. However, they are similar to RankSVM as they are regression based and would require input features with explicit values.

In order to learn the parameters of my new opportunity cost model model, found in Equation 3.16, stochastic gradient descent [32] is employed on the model. Stochastic gradient descent slowly learns the model based on a set of update rules. It trains each of the k features of each feature vector. For each feature, it trains over a given number of passes. On each pass, every training instance is updated according to a set update rules. For each update, a small learning rate, λ, is used. This value is usually set to a relatively small value (e.g. .001). In order to prevent over-fitting of the model and to allow for each of the k features to be significant, a regularization parameter, ρ, is added to the model. For each training instance, the error, ϵ, is found and determines to what extent each parameter will be updated. The higher value tells which is the predicted preference in the ranking order.

Each parameter, θ, of the model is updated based on the same method as shown in Equation 3.17. The parameters of my model are ϕ_u, ϕ_i, ϕ_j, and α_u. Equation 3.17 defines how any of these parameters can be updated. For each parameter, the change involves the learning rate, λ, the error, ϵ, the interaction of the parameter with respect to other parameters (done using the partial derivate), the regularization weight, ρ, and the current value of the parameter, θ. I define error based on the pairwise loss function found in Equation 3.2. This is more commonly referred to as Heaviside loss as the output of the function is always 0 or 1. The error is multiplied with the partial derivative of the model from Equation 3.16 with respect the parameter as found in the definition of stochastic gradient descent. The regularization weight is multiplied with the current value of the parameter to prevent over-fitting of the model. The difference between these values is multiplied by the learning rate to make sure the model is updated gradually.

$$\theta := \theta + \lambda \left(\epsilon \cdot \frac{\partial \hat{r}_{ui|j}}{\partial \theta} - \rho \cdot \theta \right) \tag{3.17}$$

An example of the partial derivative of the model with respect the parameter α_u is shown below is Equation 3.18.

$$\frac{\partial \hat{r}_{ui|j}}{\partial \alpha_u} = \frac{\partial \left(\phi_u \cdot \phi_i + \alpha_u \cdot [\phi_u \cdot \phi_j]\right)}{\partial \alpha_u} = \phi_u \cdot \phi_j \tag{3.18}$$

The update rules found in Table 3.3 are used in the overall update algorithm found in Algorithm 1. The algorithm takes in the set of pairwise preferences, $\mathcal{P}$, that relates items and users. This algorithm goes over each each index of the latent feature vectors from 1 to k. For each feature, it take a predetermined number of training passes. On each training pass, each preference in the set of preferences is updated according the update rules. A preference involves as user u and two items, i and j where it is known that i is preferred to j.

Table 3.3: Update Rules for Parameters of the Opportunity Cost Model

Parameter	Update Rule
α_u	$\lambda \cdot \left(\epsilon \cdot \left[\sum_{f=1}^{k} \phi_u[f] \cdot \phi_j[f]\right] - \rho \cdot \alpha_u\right)$
$\phi_u[f]$	$\lambda \cdot (\epsilon \cdot [\phi_i + \alpha_u \cdot \phi_j[f]] - \rho \cdot \phi_u[f])$
$\phi_i[f]$	$\lambda \cdot (\epsilon \cdot \phi_u[f] - \rho \cdot \phi_i[f])$
$\phi_j[f]$	$\lambda \cdot (\epsilon \cdot [\alpha_u \cdot \phi_u[f]] - \rho \cdot \phi_j[f])$

Algorithm 1 Update Algorithm for the Opportunity Cost Model

```
LEARN($\mathcal{P}$)
 1: for all $f = 1 \to k$ do
 2:     for all $pass = 1 \to passes$ do
 3:         for all $p \in \mathcal{P}$ do
 4:             $\epsilon = l(p)$
 5:             $\alpha_u += \lambda \cdot \left(\epsilon \cdot \left[\sum_{f=1}^{k} \phi_u[f] \cdot \phi_j[f]\right] - \rho \cdot \alpha_u\right)$
 6:             $\phi_u[f] += \lambda \cdot (\epsilon \cdot [\phi_i + \alpha_u \cdot \phi_j[f]] - \rho \cdot \phi_u[f])$
 7:             $\phi_i[f] += \lambda \cdot (\epsilon \cdot \phi_u[f] - \rho \cdot \phi_i[f])$
 8:             $\phi_j[f] += \lambda \cdot (\epsilon \cdot [\alpha_u \cdot \phi_u[f]] - \rho \cdot \phi_j[f])$
 9:         end for
10:     end for
11: end for
```

The time complexity of running this learning method is a function of the number of training passes, $passes$, the length of the latent feature vector, k, and the number of preferences in the training set, $|\mathcal{P}|$. This makes the time complexity of using stochastic gradient descent be $O(passes \cdot k \cdot |\mathcal{P}|)$. It is important to note that in comparison to other methods the complexity of the input data, in my case preferences denoted as $\mathcal{P}$, is larger as preferences in my case are $\mathcal{I} \times \mathcal{I} \times \mathcal{U}$ in comparison to other methods which is $\mathcal{I} \times \mathcal{U}$. In my case, items are accounted for twice because I am making pairwise comparison of item/item pairs where traditionally only ratings on user/item pairs are made.

3.5 Ranking Items for Output

After the model is learned offline using the pairwise preferences, the system will need to make predictions based on the model to make recommendations to the user. When a user comes to the system to ask for a recommendation, it will be done on a set of items. For each possible pair of items generated from the set of items, I can check the relevance values of each item given the other item and know the predicted preference using Equation 3.16. This can be done for each pair of items to derive a ranked order.

It is worth noting that it is possible that deriving a ranked order based on pairwise preferences can induce cyclical preferences (e.g. $A \succ B$, $B \succ C$, $C \succ A$). These are not desirable as they make it difficult to display a ranked order to the user. If this happens, the system would have to make a decision to display either A, B, C or C, A, B. In either case, there is one pairwise inversion being introduced based on the set of determined preferences.
Deterministically, the system should choose the order that reduces the number of inversions for the other preferences.

3.6 Alternative Method for Numerical Optimization

There are some drawbacks to using the pairwise loss function to assign the error used in stochastic gradient descent. The main problem is that the result is always either 0 or 1 which may not be good in some cases. For example, if the difference between the predicted relevance values for *Forrest Gump* and *Green Mile* is .001. In the first case, *Forrest Gump* may be predicted to be .001 higher than *Green Mile*. For example, the predicted relevance for *Forrest Gump* may be .5001 and *Green Mile* may be .5. For this, the model captures the users preference, but does not differentiate between the two very well which may be bad for the generalization of the model beyond the training data. In this case, it may be smart to update the model such that the relevance values are more spaced out. This would try to make the .5001 value for *Forrest Gump* be larger and the .5 value for *Green Mile* be smaller. In the other case where *Green Mile* is preferred to *Forrest Gump*, *Green Mile* may be .001 higher than *Forrest Gump*. In this case, the loss will be 1 and the model will be update the same way as if *Green Mile* was predicted to be .5 higher than *Forrest Gump*. In this case, it may be smart not the have loss at 1, but a smaller value because the difference between values (.5 and .5001) is negligible. Furthermore, the lack of quality for results using the other Heaviside loss method can be shown in [75] where other loss methods were explored.

The use of Heaviside loss directly correlates to the optimization goal for the output. However, it does not always produce the best results as seen in other work such as [75]. Because of this, I offer a surrogate objective that can be used to solve the same problem. The only difference with the alternative method is how error is calculated for the update rules. Previously, the error was always assigned based on the Heaviside loss of the current preference as seen in Equation 3.2. With the alternative method, each training instance is updated twice. The first update looks at the more preferred item and tries to optimize $\hat{r}_{ui|j}$ to have a value of 1. This is done by assigned the error such that I look at the current value of $\hat{r}_{ui|j}$ with respect to the value of one as seen in the below equation.

$$\epsilon = 1 - \hat{r}_{ui|j} \tag{3.19}$$

In this case, the error will be assigned the value of 1 - $\hat{r}_{ui|j}$. After the error is calculated, I again update the 4 parameters of the model, ϕ_u, ϕ_i, ϕ_j, and α, with this error value. This is done using lines 5 to 8 of Algorithm 1. After this, I consider the lesser preferred item and try $\hat{r}_{uj|i}$ to have a value of 0 which is done using the below equation.

$$\epsilon = 0 - \hat{r}_{uj|i} \tag{3.20}$$

For this, I reassign the error value to be 0 - $\hat{r}_{uj|i}$ and update the parameters of the model again with the new error term.

The time complexity of this method does not change from my previous explanation. Although more parameters are added to the model, this calculations and updates can be done in linear time.

3.7 Comparison with Related Work

In this chapter, I presented a novel opportunity cost model to handle pairwise preferences as a general form of user feedback. This model predicts a relevance value for an item based on a comparison with another item.

This chapter presented a new problem setting previously unseen in existing work. The traditional problem setting of recommender systems, as seen in Section 2.5, involves a set of items, a set of users, an a set of ratings which are given by users on items. These ratings are usually cardinal relevances values (e.g 1, 2, 3, 4, 5). Other work, such OrdRec, in Section 2.6, extend this idea by allow for ordinal relevance values (e.g. A, B, C, D, F). These values can be interpreted such that there is a relative order between each value (e.g. A is better than B, etc...). In my problem setting, I allow for an even more general form of input data in the form of pairwise preferences (e.g. item i is better than item j). These can be induced from ordinal or cardinal values, but they can also be generated otherwise by implicit feedback as seen in Table 3.2. This is more general as it does not restrict a relevance scale such as only 1 to 5 or A to F which both only have 5 relevance classes.. There may exist enough pairwise preferences to induce a total order of all items which would be up to the cardinality of the set of items number of relevance classes.

Because this is the first work to handle pairwise preferences, it requires different recommendation techniques to find the parameters of the model. Looking at the optimization goal of RSVD in Equation 2.1, it looks at minimizing the difference between the prediction relevance value and the rating provided by the user. This is case for cardinal values. OrdRec, described in Section 2.6, operates on ordinal data, but takes a different route by first having to generate a probability distribution. Because of this, OrdRec attempts to optimize to making the probability distribution rather than a predicted value. CCF, described in Section 2.8, using binary relevance and attempts to optimize by looking at the difference the relevance of the chosen item and the non-chosen items. My optimization goal, as seen in Equation 3.1, looks are directly minimize the amount of pairwise loss, a common evaluation technique for ranking. The is not seen in other techniques as they do not look at trying to optimize for ranking, even if they attempt to evaluate on ranking.

Additionally, this idea of exploring opportunity cost has only been studied by one other work, [75], which was done only which respect to feedback collected in user sessions. I generalized this approach to the case of only having two items that are being compared.

Chapter 4

CONTEXT-AWARE METHODS

4.1 Introduction

Beyond allow for pairwise preferences, there is the possibility to exploit information about the context of a user choice in terms of the offer set of items shown to a user at the time of recommendation. For example, active user may come to a movie recommender system and pose the query "*Tom Hanks actor*." The system will then find all of the movies in which Tom Hanks was an actor and then attempt to rank them for the active user. The movies constitute the offer set of movies. After the system returns these items to the user, the user will interact with the returned items accordingly. For example, the system may return the movies found in Table 3.1 and the user may have the corresponding actions. In this example, I consider three different user actions, but these could be differentiated or changed accordingly for each individual system. The possible user actions in the example are watching the movie, clicking the movie to examine details such as actors or plot, or there may be no user interaction. I can interpret this implicit user feedback to say that any movies the user watches are preferred to the movies the user only clicks. I can also say that any movies that the user watches or clicks are preferred to the movies that the user has no interaction. Our system will use the log of all of these user session interactions to train the parameters of a model to make new predictions for the active user's current session.

It is important to leverage the feedback given from users found in sessions. For example, knowing that a user choose to purchase *Forrest Gump* over a set of drama movies may tell us that the user likes Tom Hanks. However, if the user chooses *Forrest Gump* over a set of other Tom Hanks movies, it may tell us that the user prefers the genre or other actors starring in the movie. However, only one previous work, Collaborative Competitive Filtering (CCF) [75], looks at leveraging this session information to make recommendations.

As the only existing work, CCF [75], works on this type of setting is limited to binary preferences, I attempt to handle any number of levels of user preference. Binary relevance is limited as it only allows to have a set of preferred items and a set of non-preferred items. As described above, there is a possibility to generate finer grained preferences based on implicit user feedback. The challenge in extending binary user feedback is how to handle any number of levels of user preference. Binary preferences are limiting in how to interpret as they can be either cardinal (e.g. 0 or 1) or ordinal (e.g. good or not good). The challenge comes with allowing for a dynamic relevance scale that is up to the size of the offer set. This means that a total order of items could be made as the number of possible relevance values is always the number of items in the offer set. This is difficult as sometimes there may be two levels of relevance within a session (e.g. clicked, not clicked) and sometimes there by up to the number of items.

Another limitation of CFF [75] is that predictions are not context-dependent. Even though learning methods of existing work may take into consideration the context of the user's choice, no existing work on sessions look at exploiting this information. The challenge is to create a model that incorporates the context of the offer set of a session and can handle context-dependent relevance values. Currently, the only prediction model for work on sessions is the traditional combination of latent features as seen in Section 2.5. The difficulty is how to extend this model to take in the offer set as a parameter and use it to create a context-dependent relevance value for each item.

In this chapter I discuss a new problem setting relating to user sessions, an updated opportunity cost model allowing for context-dependent relevance values, and look at the associated learning techniques.

4.2 Problem Definition

In contrast to the previous problem setting, I now assume that user's preferences are collected in a specific time-frame which is called a *user session*. This changes the problem setting from the previous chapter. There is still a set of users and a set of items. However, now there is a set of user sessions which the preferences are contained. This set of user sessions is denoted as $\mathcal{S}$.

The previous chapter looked a using user feedback in terms of pairwise preferences. These pairwise preferences may be generated from the union of many different user sessions. However, when looking at feedback found in sessions, I can change my problem setting to account for a special property found with the specific case of session feedback which collect feedback that is disjoint across session. Instead of pairwise preferences, I introduce the idea of a relevance class.

Definition (Relevance Class)**.** *A relevance class is a set of items with the same perceived value to the user. These classes are ordinal in nature meaning that the relative value to the user can be determined between any two relevance classes (e.g. items that are watched or preferred to those that with no interaction).*

Relevance classes are used instead of pairwise preferences because there is a finite number of user actions that can be distinguished within user sessions (e.g. purchases, clicks, non-interaction). Instead of generating pairwise preferences based on the knowledge from these different user interactions, I just cluster them into relevance classes. This is done because for small sessions (e.g. 6 items), it is possible to generate up to 15 pairwise preferences and would be common to see 10 to 12. Note that pairwise preference can still be inferred based on relevance classes, but I treat them differently to account for the special property found with user sessions.

Using relevance classes, each session consists of a user, an offer set of items, and a set of relevance classes. A single session is denoted as s an which is denoted as $(u, \mathcal{O}, \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\})$ where u is the user who the interacted with the system during the session, $\mathcal{O}$ is the offer set of items for the session, and $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$ is a set of n relevance classes. The items in the offer set are all contained in exactly one relevance class so that $\mathcal{O}$ is equal to $\{\mathcal{C}_1 \cup \mathcal{C}_2 \cup ... \cup \mathcal{C}_n\}$. The highest relevance class is $\mathcal{C}_1$ which the items contained within this class are preferred to all other classes. This means that all items in $\mathcal{C}_i$ are preferred to those in $\mathcal{C}_{i+1}$ to $\mathcal{C}_n$ for all values of i from 1 to n - 1.. Note that the number of relevance classes, n, is session specific and will not be the same value across all sessions. This is because some sessions may have two classes (e.g. purchased and no interaction), while others may have 3 or more (e.g. purchased, clicked, and no interaction).

Using the same sample query, "*Tom Hanks actor*," and user interactions from the previous section, I look at Table 4.1 to see the relevance classes that the items would be in for this problem setting. Each relevance classes consists of items that the system does not know any pairwise relationship. For example,

Table 4.1: Session Preferences Generated from Sample Query – "*Tom Hanks actor*"

Movie	Relevance Class
Forrest Gump	$\mathcal{C}_1$
Cast Away	$\mathcal{C}_1$
Green Mile	$\mathcal{C}_2$
Catch Me If You Can	$\mathcal{C}_2$
Toy Story	$\mathcal{C}_3$
Saving Private Ryan	$\mathcal{C}_3$

since both Forrest Gump and Cast Away were purchased, the system does not differentiate between the two movies. Furthermore, I can see that both of these two movies will fall into the highest relevance class, $\mathcal{C}_1$, which is preferred to all of the other movies in the offer set.

The output of this type of session is similar to the global problem setting. Based on the active offer set, the offer set for the active user, the system wants to output a ranking order that minimizes the pairwise loss between the preferences implied by the ranking order. I look at doing this for each user and their set of session. A user is denoted as u and their set of session is denoted at $\mathcal{S}_u$. This again can be achieved by changing the parameters of the chosen learning and prediction model. I denote the set of parameters of any model as Θ. This time, I still look at pairwise loss, but this time this is done within a session, rather than over the set of user preferences. The loss within a session is shown in Equation 4.6. This is done by generated the possible preferences found within session s and using the Heaviside loss function found in Equation 4.3 which evaluates the loss for preference $p_{ui|j}$, for user u preference of item i over item j. I look at going from the most preferred relevance class, $\mathcal{C}_1$, down to the least preferred relevance class, $\mathcal{C}_n$. This is done by comparing relevance classes with two different indexes, c and d. I start with c being 1 which is for the most preferred class and d starts at $c + 1$. Since

d always starts larger than c, relevance class $\mathcal{C}_c$ is always preferred to $\mathcal{C}_d$.

$$\min_{\Theta} \sum_{u \in \mathcal{U}} \sum_{s \in \mathcal{S}_u} l_{session}(s) \tag{4.1}$$

$$l_{session}(s) = \sum_{c=1}^{n-1} \sum_{d=c+1}^{n} l(p_{ui|j}) : \forall i \in \mathcal{C}_c \forall j \in \mathcal{C}_d \tag{4.2}$$

$$l(p_{ui|j}) = \begin{cases} 0 & : \hat{r}_{ui|s} > \hat{r}_{uj|s} \\ 1 & : \hat{r}_{ui|s} \leq \hat{r}_{uj|s} \end{cases} \tag{4.3}$$

Looking back at the preferences shown in Table 4.1, I would like to talk about the set of inequalities that I would like to enforce based on this optimization goal. In Section 3.2, I generated the set of pairwise preferences based on the implicit user feedback. This was because I needed to make a relevance prediction for a particular item based on only one other item. For this problem setting, I need to generate a predicted relevance value based on a set of items. For this, I do not need to generate a full set of preferences. I attempt to make sure that the predicted relevance value for a movie is less than those values for more preferred relevance classes and greater than items in lesser preferred relevance classes. For example looking at the movie *Green Mile*, I will make sure that its predicted relevance is less than that of *Forrest Gump* and *Cast Away* while it is also greater than *Toy Story* and *Saving Private Ryan*. Note that I do not compare it with *Catch Me If You Can* as it is in the same relevance class.

4.3 Contextual Opportunity Cost Model

I further extend the opportunity cost model to handle the context of a user session with many items. Below is the extend opportunity cost model. Now, I look at an *offer set* of items to try to assign a relevance value to each item. For a particular item, its relevance value is its benefit to the user and the opportunity cost of giving up the other items. Looking at Equation 4.4, the relevance value of item i is being assigned based on the remaining items in the offer set of session s which is denoted as $j \in \mathcal{O} \setminus \{i\}$. This is because I am trying to assign a relevance value to an item assuming that the user is only trying to value this item and would have to give up the other items. The benefit of item i is denoted as b_{ui} and the opportunity cost of item j is denoted as c_{uj}.

$$\hat{r}_{ui|s} = b_{ui} + \sum_{j \in \mathcal{O} \setminus \{i\}} c_{uj} \tag{4.4}$$

Again, I extend this model to allow for each item and user to have a vector of features as seen below. The sum of the opportunity costs are averaged to allow for differing sized offer sets across sessions. If the average is not taken, then the opportunity cost for session containing 10 items would be drastically greater than those with only 4 items. This technique was also used in [75].

$$\hat{r}_{ui|s} = \phi_u \cdot \phi_i + \alpha_u \cdot \left[\frac{1}{|\mathcal{O}| - 1} \sum_{j \in \mathcal{O} \setminus \{i\}} \phi_u \cdot \phi_j \right] \tag{4.5}$$

The key advantages of this model are that it directly accounts for the context of the offer set. This means it will generate a session-specific relevance value for each item in a session. If an item appears in multiple sessions for the same user, then the item will be given a new relevance value for each session based on the other items in the session.

4.4 Learning the Model

As in Section 3.4, stochastic gradient descent is employed on the model to learn its parameters, ϕ_u, ϕ_i, ϕ_j, and α_u. The update rules for the model presented in the previous section are given in Table 4.2 where λ is the learning rate, ϵ is the error, and ρ is the regularization rate.

The error, ϵ, for each update is assigned as in Equation 4.6. I iterate from the most relevant class down to the class of the current item i which is

Table 4.2: Update Rules for Parameters of the Contextual Opportunity Cost Model

Parameter	Update Rule
α_u	$\lambda \cdot \left(\epsilon \cdot \left[\frac{1}{\lvert\mathcal{O}\rvert-1} \sum_{j \in \mathcal{O} \setminus \{i\}} \sum_{f=1}^{k} \phi_u[k] \cdot \phi_j[k]\right] - \rho \cdot \alpha_u\right)$
$\phi_u[k]$	$\lambda \cdot \left(\epsilon \cdot \left[\phi_i + \frac{\alpha_u}{\lvert\mathcal{O}\rvert-1} \sum_{j \in \mathcal{O} \setminus \{i\}} \phi_j[k]\right] - \rho \cdot \phi_u[k]\right)$
$\phi_i[k]$	$\lambda \cdot (\epsilon \cdot \phi_u[k] - \rho \cdot \phi_i[k])$
$\phi_j[k]$	$\lambda \cdot \left(\epsilon \cdot \left[\frac{\alpha_u}{\lvert\mathcal{O}\rvert-1} \cdot \phi_u[k]\right] - \rho \cdot \phi_j[f]\right)$

denoted as $class(i)$. I then go from the next class to the lowest preferred class. For each of the other classes, I compare the Heaviside loss, as seen in Equation 4.6, with the current item i and each item in the other class. If the errors occur with more relevant classes, then the value of this item need to go down so the error will be negative. If the errors occur with less relevant classes, then the value of the item will need to get larger so the error will be a positive value.

$$\epsilon = -\sum_{c=1}^{class(i)-1} \sum_{j \in \mathcal{C}_c} l(p_{ui|j}) + \sum_{c=class(i)+1}^{n} \sum_{j \in \mathcal{C}_c} l(p_{ui|j}) \tag{4.6}$$

Looking at an example using the sample relevance values in Table 4.3, I can make a sample error calculation for *Catch Me If You Can*.

$$\epsilon = -(0+0) + (1+0) = 1 \tag{4.7}$$

Catch Me If You Can will first be compared with both *Forrest Gump* and *Cast Away*. Since its relevance value is lower than both of the other movies relevance values, both loss values will be 0. Next, it will be compared with *Toy Story* and *Saving Private Ryan*. Since the relevance value of *Toy Story* is greater than *Catch Me If You Can*, the loss is 1. This results in a total error of 1. This means that the value of *Catch Me If You Can* needs to be larger in order to

Table 4.3: Sample Relevance Values

Class	Movie	Relevance
$\mathcal{C}_1$	Forrest Gump	.87
$\mathcal{C}_1$	Cast Away	.96
$\mathcal{C}_2$	Green Mile	.57
$\mathcal{C}_2$	Catch Me If You Can	.32
$\mathcal{C}_3$	Toy Story	.44
$\mathcal{C}_3$	Saving Private Ryan	.31

become greater than *Toy Story*. Likewise, when updating *Toy Story*, the error would be -1 which would try to make *Toy Story* have a lower relevance value.

The time complexity of running this learning method is again function of the number of training passes, $passes$ and the length of the latent feature vector, k. However, I now look at user sessions instead of preferences. This changes the previous time complexity analysis as this method is based on the size of the offer set, $|\mathcal{O}|$, and the number of items, $|\mathcal{I}|$. This makes the time complexity of using stochastic gradient descent be $O(passes \cdot k \cdot |\mathcal{O}| \cdot |\mathcal{I}|)$. This is better than the previous method where the set of preferences was $|\mathcal{I}^2|$ whereas this method is only $|\mathcal{O}| \cdot |\mathcal{I}|$ as the offer set size is much smaller than the number of items in the system.

4.5 Predictions for the Active Session

Predictions for the active session are made when a user comes to the system to ask for a recommendation. Based on the set of items the user wants a recommendations, the system can make calculations for the prediction relevance values according to the model presented in the previous sections. Items are then ranked based on their predicted relevance.

For example we can look at the sample benefit, $\phi_u \cdot \phi_i$, values in Table 4.4. Assume that we want to make a prediction for the value of *Forrest Gump* based on the offer set of items in Table 4.4 with an opportunity cost

Table 4.4: Sample Benefit Values

Movie	Benefit
Forrest Gump	.5
Cast Away	.4
Green Mile	.3
Catch Me If You Can	.3
Toy Story	.2
Saving Private Ryan	.1

discount factor, α_u, of .3. The sample calculation is done in Equation 4.8. The prediction relevance value, $\hat{r}_{ui|s}$, is the benefit of *Forrest Gump* plus the product of the discount factor and the sum of the other item's benefits.

$$\hat{r}_{ui|s} = .5 + .3 \cdot \frac{1}{5} \cdot (.4 + .3 + .3 + .2 + .1) = .578 \tag{4.8}$$

This can also be shown for the movie with the lowest benefit, *Saving Private Ryan*, for comparison as shown below.

$$\hat{r}_{ui|s} = .1 + .3 \cdot \frac{1}{5} \cdot (.5 + .4 + .3 + .3 + .2) = .202. \tag{4.9}$$

Note that in some cases, it would be sufficient to rank items solely based on the benefit to the user as it would produce the same ranked order. However, there are other cases where larger values of α_u would not yield the same ranking order.

4.6 Alternative Method for Numerical Optimization by Relevance Classes

Similar to the previous chapter, I can create surrogate objectives to solve the same problem. Using a surrogate objective in this case has the same motivation as seen in Section 3.6 which is based on the problems of using Heaviside loss. This alternative treats the same relevance class as equals and attempts to optimize each of their relevance values to the same value.

I could use a simply objective function where each relevance class is equidistant from each other. However, user's do not value each relevance class in the same manner. That is to say that the different between the most relevant class and the next most relevant class may be different for different users. Because of this, I introduce a new set of parameters for each user.

This new parameters of this approach determine the value that each relevance class will be optimized. That is to say I attempt to optimize the prediction value of each item in the same relevance class to be the same value. This parameter is denoted as $\delta_{u,i,j}$ which is for user u and the distance between class i and class j. Items in the most relevant class, $\mathcal{C}_1$ are always optimized to the value of 1. This is done to establish a scale in which the optimization is done. Item in lower classes get assigned an optimization value according to the below equation. The prediction relevance value for item i, $\hat{r}_{ui|\mathcal{S}}$, should be equal to the right side of the equation. The numerator defines the distance between the most relevance class and the class of item i, denoted as $class(i)$. The denominator is the distance between the most relevant class and the lower relevance class. The combination of these two gives the relative distance for the relevance class that i belongs which is on the scale of 0 to 1. Again, this scale is arbitrarily chosen to have some scale for a basis of optimization. The below equation is related to Equation 4.5 as they share a common predicted value, $\hat{r}_{ui|s}$. That is to say that I want the prediction relevance value based on the parameters of $\hat{r}_{ui|s}$ to equal to optimization goal value as define by the parameters of the below equation.

$$\hat{r}_{ui|s} = 1 - \frac{\sum_{c=1}^{class(i)} \delta_{u,c,c+1}}{\sum_{c=1}^{n-1} \delta_{u,c,c+1}} \tag{4.10}$$

For example, I may want to find the optimization goal value for *Green Mile* which is in $\mathcal{C}_2$. A visual example of how these δ values are used can be seen in

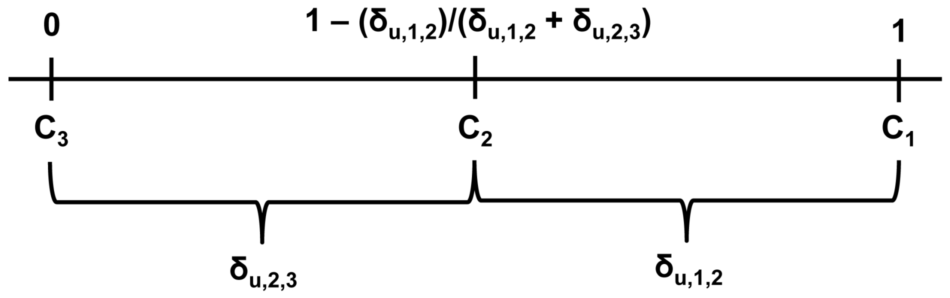


Figure 4.1: Sample for 3 Relevance Classes

Figure 4.1. For this, $\delta_{u,1,2}$ may equal 1.15 and $\delta_{u,2,3}$ may equal .95. In this case, all items in relevance class $\mathcal{C}_2$, including *Green Mile*, would be optimized to $1 - \frac{1.5}{1.5+1}$ which is equal to .4. As always, all items in relevance class $\mathcal{C}_1$ would be optimized to 1 and since $n = 3$, all items in relevance class $\mathcal{C}_3$ would be optimized to 0 as the numerator and denominator would be equal.

The main change for using this type of numerical optimization with the previous learning method is how the error term is calculated. The error is now based on how far the optimization goal is away from the current predicted relevance value. For this, the error term, ϵ, is assigned as follows:

$$\epsilon = 0 - \left(\hat{r}_{ui|s} - \left[1 - \frac{\sum_{c=1}^{class(i)} \delta_{u,c,c+1}}{\sum_{c=1}^{n-1} \delta_{u,c,c+1}} \right] \right) \tag{4.11}$$

When using this method, this error replaces the error based on Heaviside loss as seen in Equation 4.6. This is assigned because the ideal value for the difference between these the current prediction and the optimization goal is 0. So I compare this value with the difference between the numerical optimization goal and the current predicted relevance value for the model.

Along with using the other set of update rules found in Table 4.2, there needs to be a new update rule for how to update each $\delta_{u,*,*}$ in the model. Below is the update rule for doing this type of update. Note that instead of adding to the current value I subtract. This is because in the error term, the optimization goal value is being subtracted from the current prediction for the relevance value, $\hat{r}_{ui|\mathcal{S}}$, as seen in Equation 4.11.

$$\delta_{u,i,i+1} := \delta_{u,i,i+1} - \lambda \cdot \left(\epsilon \cdot \frac{\partial \hat{r}_{u,i|\mathcal{S}}}{\partial \delta_{u,i,i+1}} - \rho \cdot \delta_{u,i,i+1} \right) \tag{4.12}$$

$$\frac{\partial \hat{r}_{u,i|s}}{\partial \delta_{u,i,i+1}} = \frac{\sum_{c=1}^{n-1} \delta_{u,c,c+1} - \sum_{c=1}^{class(i)} \delta_{u,c,c+1}}{\left(\sum_{c=1}^{n-1} \delta_{u,c,c+1} \right)^2} \tag{4.13}$$

4.7 Alternative Method for Numerical Optimization by Item/Item Pairs

The second alternative does not treat all items in the same relevance class as equals and allows for each item to be optimized to a different value. This is because I am trying to work with pairwise preferences which does not enforce optimizing all items in a single relevance class to be the same value. This methods looks at interpreting preferences as a graph where a directed edge between two nodes (or items) can been seen as preference. I can construct a graph based on the relevance classes where edges are only found between adjacent relevance classes and following a path between any two items can show their relative preference.

I again introduce a new set of parameters for this technique. The new parameter is denoted as $\delta_{u,i,j}$ where its value is for user u between items i and j. This parameter represents the difference in value between two items. The below equation equation shows how these parameters can be used to find the optimization value for a given item i. Again, this is done on a scale of 0 to 1 which is done arbitrarily to establish a scale for the optimization. This approach again looks from going from the most relevant class down to the relevance class of item i. It does so finding the average distance for paths from $\mathcal{C}_1$ down to the class before the class of i. However, instead of taking the average of all possible path which is exponential in time complexity, it only looks at finding the average distance to the relevance class before the class of the item we are trying to optimize denoted as $class(i)$. Then it finds the average distance between all items in the class preferred just more than item i to item i. For the denominator, it is the average distance to the least relevant class.

$$\hat{r}_{ui|s} = 1 - \frac{\sum_{c=1}^{class(i)-2} \frac{1}{|\mathcal{C}_c \times \mathcal{C}_{c+1}|} \sum_{a \in \mathcal{C}_c, b \in \mathcal{C}_{c+1}} \delta_{u,a,b} + \frac{1}{|\mathcal{C}_{class(i)-1}|} \sum_{a \in \mathcal{C}_{class(i)-1}} \delta_{u,a,i}}{\sum_{c=1}^{n-1} \frac{1}{|\mathcal{C}_c \times \mathcal{C}_{c+1}|} \sum_{a \in \mathcal{C}_c, b \in \mathcal{C}_{c+1}} \delta_{u,a,b}} \tag{4.14}$$

A visual representation of this method and its parameters can be seen in Figure 4.2. In this figure, there are four relevance classes. To see how Equation 4.14 works, I offer the sample parameters in Table 4.5 to show an example calculation. The vertical column represents the first items in the subscript of the δ and the horizontal column represents the second. I use a ? to denote values that are not shown in the sample figure. Note that these parameters would have a value, but I do not show them to make it easier to find the values needed for the calculation. Also note that all – values are for those combination where no parameters existing as there is no distance between an item and itself. Below I show the sample calculation for the value of $\hat{r}_{ui_5|\mathcal{S}}$.

$$\hat{r}_{ui_5|s} = 1 - \frac{[\frac{1}{2}(\delta_{ui_1i_2} + \delta_{ui_1i_3})] + [\frac{1}{2}(\delta_{ui_2i_5} + \delta_{ui_3i_5})]}{\frac{1}{2}(\delta_{ui_1i_2} + \delta_{ui_1i_3}) + \frac{1}{4}(\delta_{ui_2i_5} + \delta_{ui_3i_5} + \delta_{ui_2i_4} + \delta_{ui_3i_4}) + \frac{1}{2}(\delta_{ui_4i_5} + \delta_{ui_5i_6})} \tag{4.15}$$

$$\hat{r}_{ui_5|s} = 1 - \frac{[\frac{1}{2}(.9 + .8)] + [\frac{1}{2}(.7 + .9)]}{\frac{1}{2}(.9 + .8) + \frac{1}{4}(.7 + 1.1 + .9 + 1) + \frac{1}{2}(.6 + .9)} \approx 0.653 \tag{4.16}$$

Table 4.5: Sample Delta Values for Figure 4.2

	i_1	i_2	i_3	i_4	i_5	i_6
i_1	–	.9	. 8	?	?	?
i_2	?	–	?	?	.7	1.1
i_3	?	?	–	?	.9	1
i_4	?	?	?	–	?	.6
i_5	?	?	?	?	–	.9
i_6	?	?	?	?	?	–

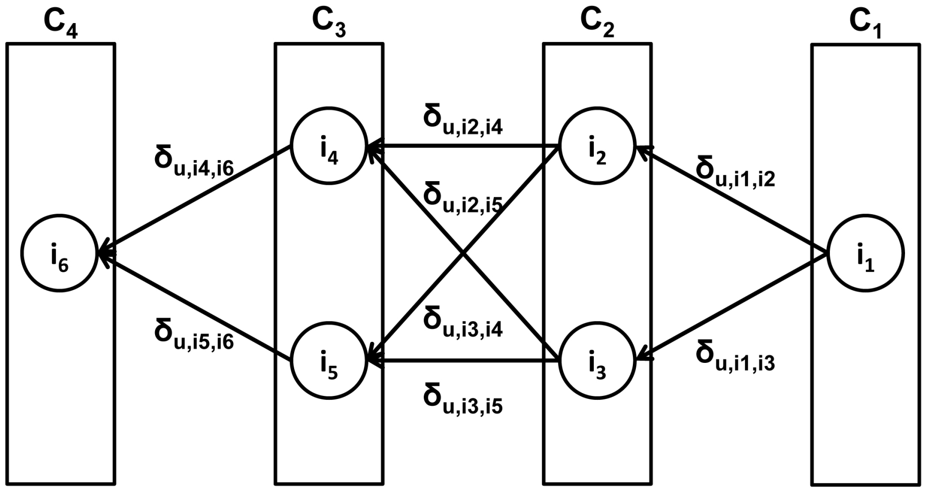


Figure 4.2: Sample for 4 Relevance Classes

4.8 Comparison with Related Work

In this chapter, I presented an extension to my novel opportunity cost model to handle user feedback generated from user session. This model predicts a context-dependent relevance value for an item based on an offer set of items given to the user.

First, I would like to discuss how session are different from other types of context enough to make it difficult to use those techniques. Traditional forms of context are time, location, or weather. I could use a technique similar to that of the user profiling discussed in Section 2.7. However, in my consideration of context, there are many values than just the number of days in a week or types of weather. I would have to enumerate all of the possible offer set combinations which would be exponential in size in relationship to the number of items. This would makes this method undesirable and not efficient.

Sessions have only been studied in one other existing work, CCF [75]. This work was limiting as it only allow for binary relevance feedback with only one item in the offer set being the most preferred item. I present a problem setting allow for any number of relevance classes up to the size of the offer set. It also does not restrict the most relevant class to only one item.

Since I change the problem setting of CCF, it required changes to the learning techniques. Looking at Equation 2.3 which is the optimization goal for CCF, I can see that it is strongly tied to to only allowing for one item in the most relevant class and makes the assumption that all other items are in the lesser preferred class. I can see this as it clearly differentiates the chosen, or most relevant item, i^*. I could make the change of having more than one item in the most preferred relevance class by taking the summation of the items not-chosen and subtracting the summation of the chosen items. However, I want to allow for a dynamic number of relevance classes which does not limit it to only two type of items. This means I cannot make a simple change in Equation 2.3 to work for more relevance classes.

Additionally, I differ from CCF in terms our prediction model. CCF uses the same prediction model as RSVD, seen in Section 2.5. This does not allow for context-independent relevance values as the model only considers one item at a time. When only considering one item at a time, it is impossible to create a context-dependent relevance value because it does not look at the context of the offer set. I present a contextual model that incorporates the context of the offer set to provide a context-dependent relevance value.

Chapter 5

EXPECTED DISCOUNTED RANK CORRELATION

As mentioned in Section 2.9, evaluation of recommender systems is an important issue. My work specifically look at accuracy with an emphasis on ranking. Most of the existing work looks at improving the prediction accuracy which means optimizing to an explicit relevance value. However, there are many times when only implicit feedback is collected and in these cases, existing measures and metrics fail to work. In these cases, rank accuracy measures and metrics can be used. However, some of them require information that might not also be provided using implicit feedback such as nDCG [29] needing explicit relevance values.

I propose *expected discounted rank correlation (EDRC)* [2] to measure the similarity between two sets of pairwise preferences. EDRC is a weighted measure that handles partially ordered lists derived from implicit user feedback based on the classifications seen in Section 2.10. Pairwise preferences are a generalize form of user input that can be easily generated from implicit user feedback or based on cardinal or ordinal feedback. When looking at pairwise preferences as ground truth, a common question would be how to evaluation such data.

Given a set of ground truth pairwise preferences from the user $\mathcal{G}$, and a set of predicted pairwise preferences output by the system $\hat{\mathcal{G}}$, EDRC calculates the expected correlation the two sets. Note that I may have user preferences on a large set of items based on many different user sessions. However, I only consider preferences relating items currently recommended by the system. That is, the set of items in $\mathcal{G}$ and $\hat{\mathcal{G}}$ are the same. Please note that it is possible based on historical preferences, that cycles may exist (e.g. $A \succ B$, $B \succ C$, $C \succ A$.. This is the case for the global, or pairwise preference, problem setting in Chapter 3. However, in the session problem setting, presented in Chapter 4, cycles are not possible within an individual session which would be at the level evaluation is conducted. Therefore, my method does not handle the case of cycles in user's preferences.

Similar to AP correlation and nDCG, EDRC emphasizes preserving the order of the user's most preferred items and enforcing a smaller penalty for less preferred items. Different from nDCG whose ground truth is a set of relevance scores, the ground truth supported by EDRC is a set of pairwise preferences. Different from AP correlation which requires complete pairwise preferences, EDRC allows incomplete pairwise preferences.

Two challenges must be addressed when thinking about how to evaluate rank accuracy based on incomplete pairwise preferences. How does one assign a rank to an item when a total list cannot be constructed? In Equation 2.10, $index(i)$ is the rank index in the list, but when a total order is unknown, such as the sample data in Table 5.1, a new method is needed to give a rank index. Second, how does one consider the cases where a user's preference between items is unknown? In that example, I don't know user's preference between items A and B. How to evaluate a system that makes a predication $A \succ B$? Next, I look at the first problem.

Assigning Rank and Computing Weight. In the spirit of discounted gain, I want to give different *discount* (weight) for different items. If I know the rank of an item, $R(v)$, in the ground truth, I may set the weight linearly, logarithmically, or exponentially. However, the problem is how to compute the rank of an item given incomplete pairwise preferences. From a set of pairwise

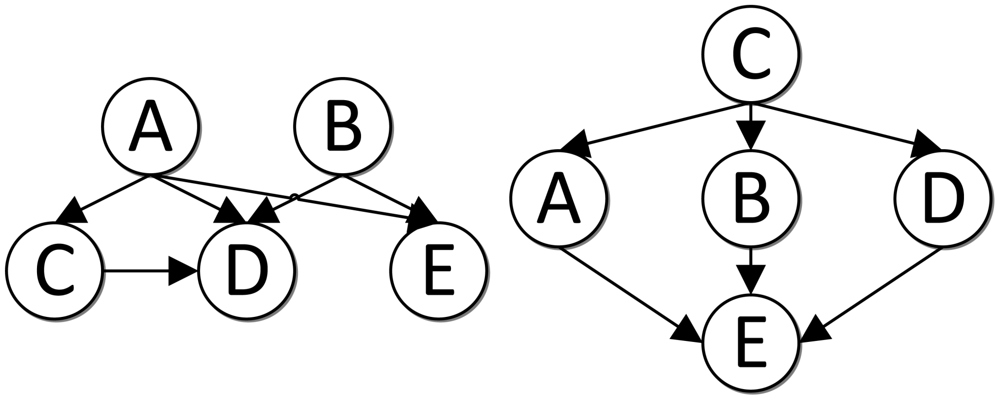


Figure 5.1: Topological Order based on Ground Truth (left) and System Prediction (right) in Table 5.1

Table 5.1: Example Pairwise Preference Data

	Preferences
Ground Truth	$A \succ C$, $A \succ D$, $A \succ E$, $C \succ D$, $B \succ D$
Prediction	$C \succ A$, $C \succ B$, $C \succ D$ $A \succ E$, $B \succ E$, $D \succ E$

preferences as ground truth, I first derive a topological order among the items. Consider a graph where a vertex represents an item, and a directed edge represents a pairwise preference. The item corresponding to the source vertex is preferred to the item corresponding to the target vertex. In the following discussion, I use item and vertex interchangeably.

For example, the ground truth in Table 2.4 can be represented by the graph in Figure 5.1, where the fact that item A is preferred to item C is shown by a directed edge from vertices A to C. In this graph, an item is preferred to any item reachable following a directed path. For example, A is preferred to both C and D in the ground truth of Table 5.1.

Now I discuss how to leverage the graph to compute item rank. I initiate the rank of every vertex to be 1 and traverse the graph beginning for the start nodes. Whenever I follow an edge from a vertex u to v, I update v's rank to

Algorithm 2 Setting Rank Value

```
SETRANKS($\mathcal{G}$)
1: $\mathcal{S}$ = all nodes in $\mathcal{G}$ without an incoming edge
2: for all $v \in \mathcal{S}$ do
3:     $R(v) \leftarrow$ 1; VISIT($v$, $1$)
4: end for
VISIT($v$, $rank_{in}$)
1: $R(v) \leftarrow \max(R(v), rank_{in} + 1)$
2: for all $w \in OUT(v, \mathcal{G})$ do
3:     VISIT($w$, $R(v)$)
4: end for
```

$max(R(v), R(u) + 1)$. I chose maximum because choosing minimum does not guarenteed that lesser preferred item have lower rank than more preferred items. Note that here I consider every user specified pairwise preference as equally important, and is assigned a unit weight of 1. I allow the rank of a vertex to weight the preference. Thus I keep the maximum score of node among the different paths that can reach this node from a start node. The procedure is defined in the procedure SETRANKS($\mathcal{G}$) in Algorithm 2.

The ranks for the items in the graph of Figure 5.1 are as follows: $R(A)$ = 1, $R(B)$ = 1, $R(C)$ = 2, $R(D)$ = 3 and $R(E)$ = 2. As I can see, nodes A and B have the same rank, since I don't know user's preferences between the two.

Then I define the discount term, $D(v)$, which can take various forms depending on what type of discount is desired. For example, for a simple linear discount, $D(v) = R(v)$. For exponential discount, $D(v) = 2^{R(v)}$ and for logarithmic discount, $D(v) = log_2(1 + R(v))$.

Handling Unknown Preferences between Items. Another problem is computing the *score*, $C(v)$, of an item in the system's output in the presence of incomplete pairwise preferences in the ground truth. I propose the following formula to define the score of an item where $OUT(v, \mathcal{G}^+)$ is the set of outgoing edges from v in the transitive closure, $\mathcal{G}^+$, of $\mathcal{G}$, and W, the set of items that are more preferred or have an unknown relationship with v where $W = V(\mathcal{G}) \setminus [\{v\} \cup OUT(v, \mathcal{G}^+)]$. EP checks for the *expected score* regarding the relationship between v and all items in W.

$$C(v) = \sum_{w \in W} EP(v, w) \tag{5.1}$$

For a pair of items (v, w), suppose v preferred over w, that is, $v \succ w$ in $\mathcal{G}$. There are three cases. I have $v \succ w$ in $\hat{\mathcal{G}}$. In this case, $EP(v, w)$ = 1. The second case, I have $w \succ v$ in $\hat{\mathcal{G}}$. Since the system prediction contradicts to the ground truth, I have $EP(v, w)$ = 0. The third case, the system cannot predict a preference between v and w. Then I need to compute the expected score. By default I may assume there is 50% likelihood for $v \succ w$ and 50% likelihood for $w \succ v$. I then let $EP(v, w)$ = .5. Alternatively, I may have a more accurate likelihood estimation based on collaborative filtering. Assuming I have a set of equally similar users, if 70% of the users have $v \succ w$ and 30% of similar users have $w \succ v$, then the expected score of $v \succ w$ is 0.7. For example, below are samples for C and EP based on Table 5.1.

$C(C) = EP(C, A) + EP(C, B) + EP(C, E) = 0 + .5 + .5 = 1$

$C(D) = EP(D, A) + EP(D, B) + EP(D, C) + EP(D, E) = .5 + .5 + 1 + .5 = 2.5$

$C(E) = EP(E, A) + EP(E, B) + EP(E, C) + EP(E, D) = 1 + 1 + .5 + .5 = 3$

Putting Things Together. Based on the discussion of how to compute a score for an item, $C(v)$, and the discount (weight) of an item, $D(v)$, I now put these together for an evaluation measure EDRC. I denote the set of all vertices in $\mathcal{G}$ without an incoming edge as $\mathcal{S}$.

$$EDRC(\mathcal{G}, \hat{\mathcal{G}}) = \frac{2}{Z} \cdot \left[\sum_{v \in V(\mathcal{G}) \setminus \mathcal{S}} \frac{C(v)}{D(v)} \right] - 1 \tag{5.2}$$

Here, Z is a normalization factor to ensure the value is between +1 and -1.

$$Z = \sum_{v \in V(\mathcal{G}) \setminus \mathcal{S}} \frac{|W|}{R(v)} \tag{5.3}$$

Considering the example data in Table 5.1, I now show how to put together the sample calculation for EDRC using a linear discount method.

$$EDRC(\mathcal{G}, \hat{\mathcal{G}}) = \frac{2}{\frac{29}{6}} \cdot \left[\frac{1}{2} + \frac{2.5}{3} + \frac{3}{2} \right] - 1 = \frac{5}{29} \tag{5.4}$$

When both the ground truth and prediction is a complete set of pairwise preferences and the discount term is $D(v) = R(v) - 1$, the values for EDRC and AP correlation will be the same.

Chapter 6

EXPERIMENTS

6.1 Introduction

I conduct experiments on our methods to show their effectiveness. I run experiments on the different problems settings in Chapters 3 and 4 along with the various methods discussed in each problem setting. I choose to evaluated on rank-oriented evaluation techniques. These techniques include pairwise loss, normalized discounted cumulative gain (nDCG), and my new measure, expected discounted rank correlation (EDRC). The remainder of this chapter discusses our datasets, the evaluation techniques, and our experimental results.

6.2 Datasets

I use three different data sets, two for the first problem setting and one for the second problem setting. The first two datasets that I look at are MovieLens 100K and are MovieLens 1M [21]. The other dataset is based on a user study I conducted using Amazon Mechanical Turk [7]. I conducted my own user study for the session problem setting because not publicly available datasets contain the session information. Publicly available datasets only contain rating information, but lack any information regarding offer sets. Since my methods needs both of these pieces of information, I decided to conduct a user study to gather this information. Below are descriptions of both of these datasets.

Please note that although I choose to use a publicly available dataset that consists solely of cardinal ratings, this does not imply that this is the only way to result in pairwise preferences. Pairwise preferences may be the result of implicit user feedback, as discussed in Section 3.2. I only generate pairwise preferences based on cardinal ratings because there are not publicly available datasets consisting of pairwise preference data.

MovieLens 100K

The MovieLens 100K dataset is a publicly available dataset from the GroupLens research group from the University of Minnesota. MovieLens is a movie recommendation engine freely available for public use. This dataset contains 100,000 ratings that are whole numbers on a scale of 1 to 5 from 943 users on 1682 movies. Each user in the dataset has rated at least 20 movies.

MovieLens 1M

The MovieLens 1M dataset is a publicly available dataset also from the GroupLens research group from the University of Minnesota. This dataset contains 1,000,209 ratings that are whole numbers on a scale of 1 to 5 from 6,040 users on 3952 movies. Each user in the dataset has rated at least 20 movies.

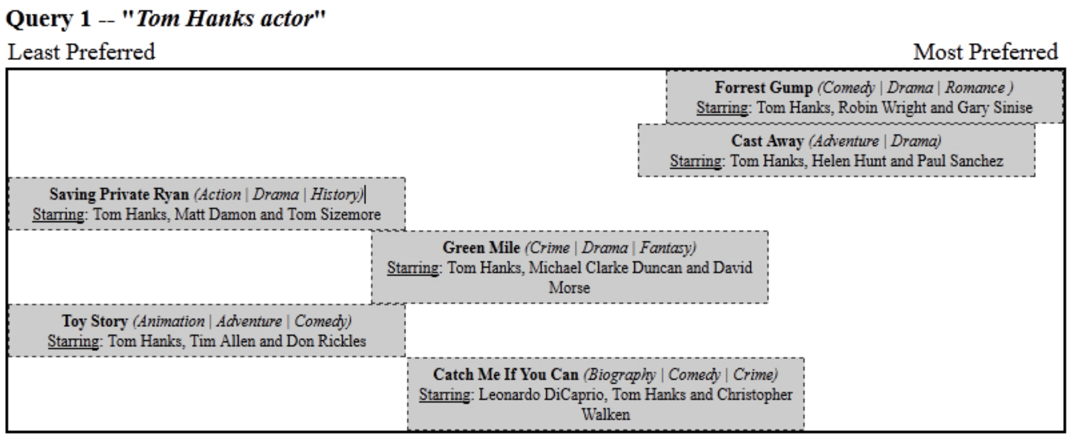
Amazon Mechanical Turk

Our dataset was a user study I conducted using Amazon Mechanical Turk. Our goal was to collect information to interpret as log data that would be implicitly gather from users in a recommender system. I collected results from 100 users. Each user was presented the same 5 queries and each query contained a set of 6 movies. Users were asked to give their preferences on the movies based on the query by moving the movies horizontally on the screen. A screen shot of query 1, "*Tom Hanks actor*", can be seen in Figure 6.1. This method was chosen as it would be difficult to build a non-commercial system that could be used to determine the most relevant movies because I could no allow users to purchase or view the movies. Additionally, I did not want to design a user study that only asked a user if they would watch or click on a movie. This would limit the number of different user interactions and would not show how well our methods can handle varying sizes of ordinal relevance classes. The directions given to users along with the 5 queries and movies can be found in the Appendices.

The queries used in the study were chosen from a list of sample queries provided by students a database management course for a course assignment. The chosen queries were selected for their ability to display movies that the majority of users would be familiar. Choosing queries that display movies that users would be familiar with is important as otherwise users would not be able to give accurate and usable feedback. The queries were issued to IMDB and then sorted based on the popularity of the movie. Popularity was defined by the number of ratings a movie received on IMDB. The top 6 movies in terms of popularity were chosen. For each movie, I found the genres, as defined by IMDB, and the first 3 actors listed for the movie. The movie name, genres and actors were displayed to the user in the study. I felt this information is sufficient for a user to be able to give their feedback on a movie.

Interpreting the Amazon Mechanical Turk User Study

Since the user study did not directly ask for the user's pairwise preferences, I must interpret these user preferences by pre-processing the data. Since this data was used to evaluate the session problem setting, results need to be clustered into relevance classes. For this, I directly use the data collected from the user study. Each movie was dragged by a user to a certain point on the



Query 1 -- "*Tom Hanks actor*"

Least Preferred | Most Preferred

Forrest Gump *(Comedy | Drama | Romance)*
Starring: Tom Hanks, Robin Wright and Gary Sinise

Cast Away *(Adventure | Drama)*
Starring: Tom Hanks, Helen Hunt and Paul Sanchez

Saving Private Ryan *(Action | Drama | History)*
Starring: Tom Hanks, Matt Damon and Tom Sizemore

Green Mile *(Crime | Drama | Fantasy)*
Starring: Tom Hanks, Michael Clarke Duncan and David Morse

Toy Story *(Animation | Adventure | Comedy)*
Starring: Tom Hanks, Tim Allen and Don Rickles

Catch Me If You Can *(Biography | Comedy | Crime)*
Starring: Leonardo DiCaprio, Tom Hanks and Christopher Walken

Figure 6.1: User Study Screenshot for Query 1

screen relative to other movies and can be assigned a value associated with the x-position on the screen. I denote a movie as m and the x-position as $x(m)$. Within each session, I normalized the x-position values to be on a scale of 0 to 1 to allow for fair comparisons between each user session. This normalization is done to account for varying screens sizes of users. The smallest x-position was assigned the value of 0 and the largest was assigned the value of 1. All other movies will fall somewhere within this range.

Because of the imprecision of users with dragging the various movies around the screen, I generate relevance classes allowing for movies that are very close together being in the same class. In order to generate the relevance classes of the session problem setting, I used a modified version of bisecting k-means clustering algorithm to place items in relevance classes. In our approach, I do not try to find k clusters, but rather try to ensure the distance similarity within in each cluster. This is done as I do not need to guarantee that each session will always have k relevance classes. I choose to ensure similarity within clusters to have similarity values movies in the same relevance class. First, I find the minimum and maximum x-position values within the set of movies. If the span of the minimum and maximum is less than a particular delta value, I stop splitting this cluster. Otherwise, I then find the centroid, which for one-dimensional data is just the average value. I then generate a random number, z, between the maximum and minimum values. From this value z, I create two new points, one for the left side of the centroid and the other the right side. Depending on which side of the centroid the random value falls, I assign the left and right values accordingly. Whichever variable is unassigned, left or right, gets assigned the mirror image of the other with respect to the centroid. For example, if the centroid is .5 and z is equal to .25, I assign the left value to be .25 and assign the right value to be $|centroid - z|$, or $|.5 - .25| = .25$ larger than the centroid meaning right is .75. I then see whether each movie's x-position if closer to the left or right value and assign the movie to a new cluster accordingly. Then I run the same splitting technique on both the left and right clusters.

After the clusters are generated, they can easily be ordered based on the x-positions found within each cluster. The cluster with the greatest values will be assigned as the most preferred cluster $\mathcal{C}_1$. Then each cluster can be assigned in descending order.

Analysis of User Study

I tried to analyze the possible effect of the offer set in the user decision making process. To do this, I have two pairs of movies that showed up in different user sessions. The first pair of movies was *Forrest Gump* and *Cast Away* which show up for both the queries of *Tom Hanks actor* and *Robert Zemeckis director*. The second pair is *Saving Private Ryan* and *Catch Me If You Can* which show up for both the queries of *Tom Hanks actor* and *Steven Spielberg director*. I do this because it is possible that a user may prefer one movie in a given context and the other movie in another context. For example, it is possible that a user prefers *Forrest Gump* over *Cast Away* when comparing results for *Tom Hanks actor*, but they prefer *Cast Away* over *Forrest Gump* for the results of *Robert Zemeckis director*. This could be because they are thinking about Tom Hanks for the first query and evaluating his acting while in the other, they are thinking about the direction of the movie.

I notice that there were few users where this a change in preference occurred between sessions. A changed in preference between *Forrest Gump* and *Cast Away* was observed in 3 users and a changed in preference between *Saving Private Ryan* and *Catch Me If You Can* occurred in 2 users. It is worth noting that these users were not the same for both changes in preference. That is to say that there are 5 different users of the total 100 that had a change in preference between sessions. However, it is worth noting that beyond a change in preference, they were also occurrences of where a preference was held in one session and no preference was found in the other session. This occurred for 8 users for *Forrest Gump* and *Cast Away* and 6 users for *Saving Private Ryan* and *Catch Me If You Can*. It is worth noting that again, these users were not the same for both cases. That is to say that this happened to 14 different users. Additionally, only 2 of the users fell into both cases meaning there is only a 2 user overlap between the 5 users with preference reversal and the 14 with a preference in one session and no preference in the other. This means that these cases are seen in 17 users which is 17% of all users.

6.3 Evaluation Measures and Metrics

Since I are attempting to improve the rank accuracy, I choose to evaluate on rank-oriented measures and metrics. I evaluate on the three measures and metrics discussed in detail in Chapter 5: pairwise loss, nDCG, and EDRC.

Pairwise loss is chosen for evaluation as it is the loss function used in our optimization goals for both problem settings. nDCG is used as it is a weighted measure which tells how well the system can rank highly preferred items. As nDCG requires explicit relevance values, it can only be used on the MovieLens 100K dataset that has these explicit relevance values. Additionally, nDCG is used to compare with Collaborative Competitive Filtering (CCF) [75] using the same modifications laid in their evaluation. This will be discussed in detail in the subsection of Section 6.4 discussing this comparison. I use our measure, EDRC, to evaluate the session problem instead of nDCG because EDRC can handle implicit user feedback. EDRC gives the same weighted characteristic as nDCG. Further discussion of how these measures and metrics will be used is found below.

Pairwise Loss

Pairwise loss is used to evaluate both problem settings and all methods. The general optimization goal for all methods is the minimize the pairwise loss over all preferences. To evaluate this goal, I use the average loss over all preferences. For the global problem setting this evaluation is done using Equation 6.1 where $\hat{\mathcal{P}}$ is the set of predicted pairwise preferences. The loss function is the same as in Section 6.4 where the result is 1 is the preference is violated by the prediction model and 0 otherwise.

$$l(\hat{\mathcal{P}}) = \frac{1}{|\hat{\mathcal{P}}|} \sum_{p \in \hat{\mathcal{P}}} l(p) \tag{6.1}$$

The same technique can be used on the session problem, but all possible pairwise preferences must be enumerated based on the relevance classes such that any preference found in relevance class $\mathcal{C}_i$ is preferred to all items in $\mathcal{C}_{i+1}$ to $\mathcal{C}_n$ where n is the number of relevance classes in a session.

nDCG

nDCG is used to evaluate the pairwise preference problem setting and to compare with CCF. To evaluate on the pairwise preference problem setting, I can use the technique in Equation 6.2. I again look at the set of predicted pairwise preferences, $\hat{\mathcal{P}}$, but I iterated them on a per-user basis where $\mathcal{U}$ is the set of users and $\hat{\mathcal{P}}_u$ are the preferences for user u. I take the average of all nDCG calculations on a per-user basis. For each user, the items are ranked based on the preferences and this is used to find the discounted cumulative gain which is divided by the ranked list of items based on the user ratings.

$$nDCG_{global}(\hat{\mathcal{P}}) = \frac{1}{|\mathcal{U}|} \sum_{p \in \hat{\mathcal{P}}_u} nDCG(\hat{\mathcal{P}}_u) \tag{6.2}$$

EDRC

EDRC is used to evaluate the session problem setting explicit relevance values are not known. Evaluation is conducted on each user session and the average is taken on over all users and and sessions. I iterate over each user u in the set of users,$\mathcal{U}$, and each session from the user u's set of sessions, $\mathcal{S}_u$. Within each session, the set of preferences are generated based on the method described in the description for pairwise loss. EDRC is calculated based on the different between the set of known preferences, $\mathcal{P}$ and the set of predicted preferences, $\hat{\mathcal{P}}$.

$$EDRC(\mathcal{U}) = \frac{1}{|\mathcal{U}| \cdot 5} \sum_{u \in \mathcal{U}} \sum_{s \in \mathcal{S}_u} EDRC(\mathcal{P}, \hat{\mathcal{P}}) \tag{6.3}$$

6.4 Experimental Results

I ran a variety of experiments to test both the pairwise preference and session problem settings against other methods and variations of my methods.

Evaluation Methodology

For each of the experiments being ran on my methods, I adopt the same methodology. Please note that for comparison against related work, I adopt the same methodology as mentioned in the original paper. For all methods, including those used as a comparison system, I always use a 5-fold cross validation technique. I first partition the dataset into 5-folds. For the SVD-based methods (both RSVD and SVD++), the dataset consists of user ratings on items where each point is one user, one item, and the user's rating on that item. The folds for SVD-based methods consists of this type of rating. For the pairwise preference problem setting, each fold consists of preferences. To generate these preference folds, I generate all preferences based on the the folds for the SVD-based methods such that all ratings with a higher value are preferred to those of a lower value. For example, for a given user, all items rated 5 are preferred to those rated 4, 3, 2, and 1. For the session problem setting, each fold consists of one user session. These folds are randomly generated. Folds for the session problem setting are randomly shuffled to ensure each user's folds correspond to different queries. I run all methods five different times withholding each of the folds into the test set on each run. For example, for the first run, the second, third, fourth, and fifth folds make up the training set and the first fold is the test set. Our methods are used to train the models based on the training set and then evaluation is conducted on the test set.

Pairwise Preference Problem Setting Results

First, I look at the pairwise preference, or global, problem setting as seen in Chapter 3. I ran experiments to compare with related work with respect to loss, nDCG, EDRC, learning time, and prediction time on both MovieLens datasets. Additionally, I show a comparison of changing learning parameters for the learning rate and regularization rate for my model which shows these changes for loss and nDCG. The work I compare against is RSVD [19] and SVD++ [33]. For both loss and nDCG, I look at comparing varying lengths of latent feature vectors which is denoted as k. I use the best parameters for my model which is a learning rate of .002 and regularization rate of .02 unless otherwise noted. For the length of the latent feature vectors, I use 3, 6, 12, 25, 50, 100, and 200. Note, I truncate results for MovieLens 1M at 100 latent features due to the large learning time. Additionally, I use 15 training passes which is cited in literature, [19], to be enough for learning a model using stochastic gradient descent. For RSVD and SVD++, I used a learning rate of .001, a regularization rate of .02, and 15 passes per features as used in [19].

Figures 6.2, and 6.3 show the results comparing against related work for loss. I used the label of GPW for my global pairwise model with HVS used for the Heaviside loss technique and OPT for numerical optimization. Looking at the Figure 6.2, which is for evaluation on loss on the smaller dataset, I see that all methods improve slightly when increasing the number of latent features, k. It is apparent that the Heaviside loss technique performs the worst. RSVD and SVD++ also performs worse than my model with a numerical optimization. At k=200, my method performs around .04 for MovieLens 100K and MovieLens 1M.

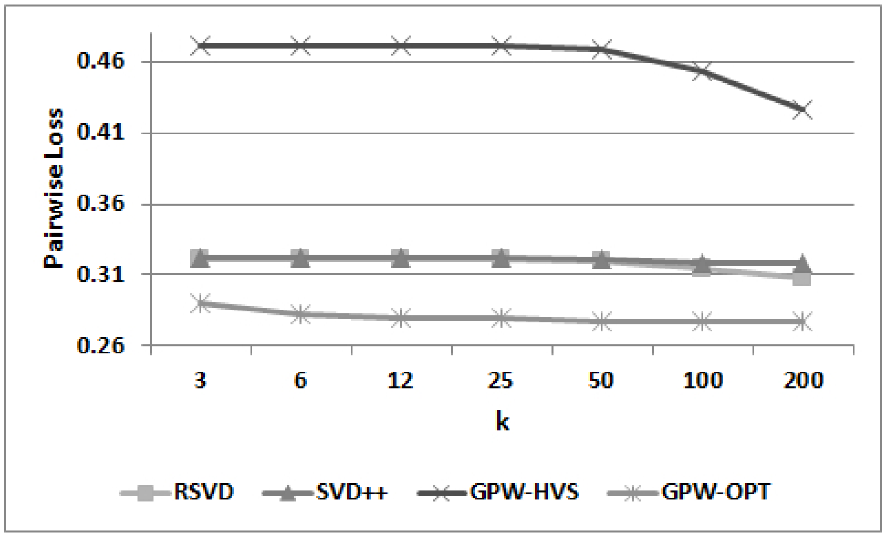


Figure 6.2: Pairwise Loss Comparison for Pairwise Preferences on MovieLens 100K

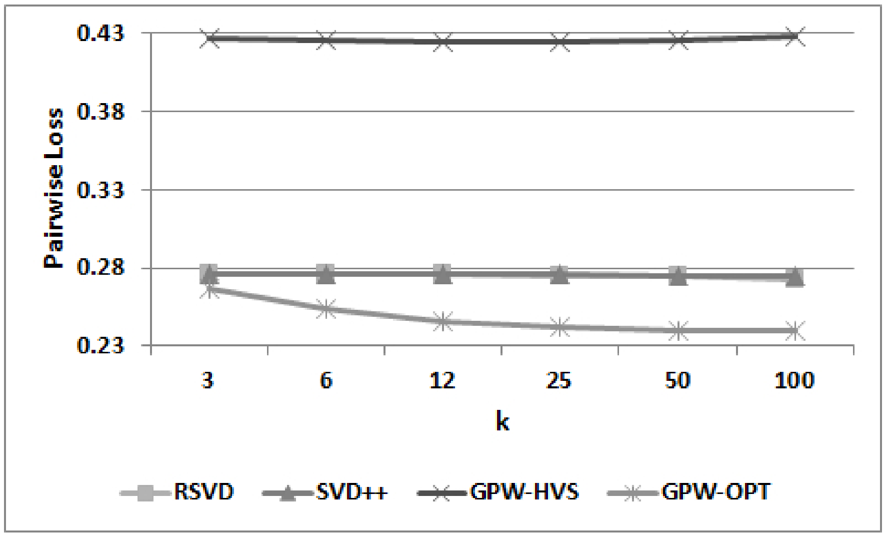


Figure 6.3: Pairwise Loss Comparison for Pairwise Preferences on MovieLens 1M

Figures 6.4, and 6.5 show the results comparing against related work for nDCG. I again use the same labels as pairwise loss for the various methods. Please note that a higher value for nDCG indicates a better quality results with 1 being the best and 0 being the worst. For these figures, I used a different learning and regularization rate in order to beat RSVD and SVD++.

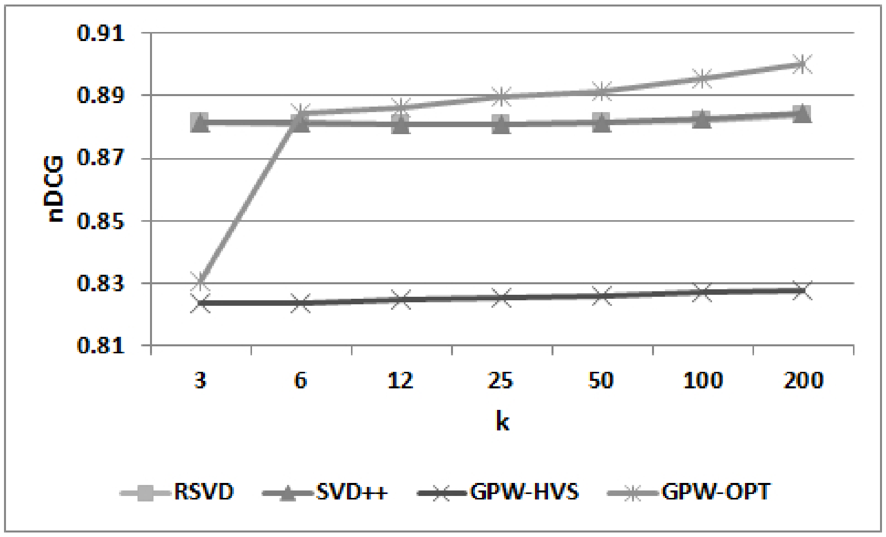


Figure 6.4: nDCG Comparison for Pairwise Preferences on MovieLens 100K

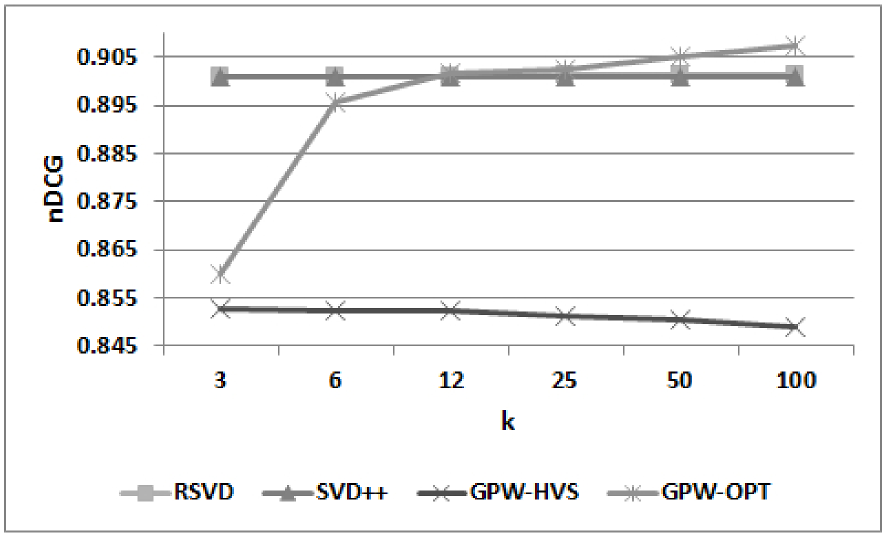


Figure 6.5: nDCG Comparison for Pairwise Preferences on MovieLens 1M

For these figures, the learning rate was .00075 and regularization rate was .014. The combination allowed me to beat the related work. Better results are also seen with increasing the number of latent features, k. For MovieLens 100K my method consistently holds a lead of around .03 all the way to k=200 where the lead is .04. For MovieLens 1M, my method takes until k=25 to match

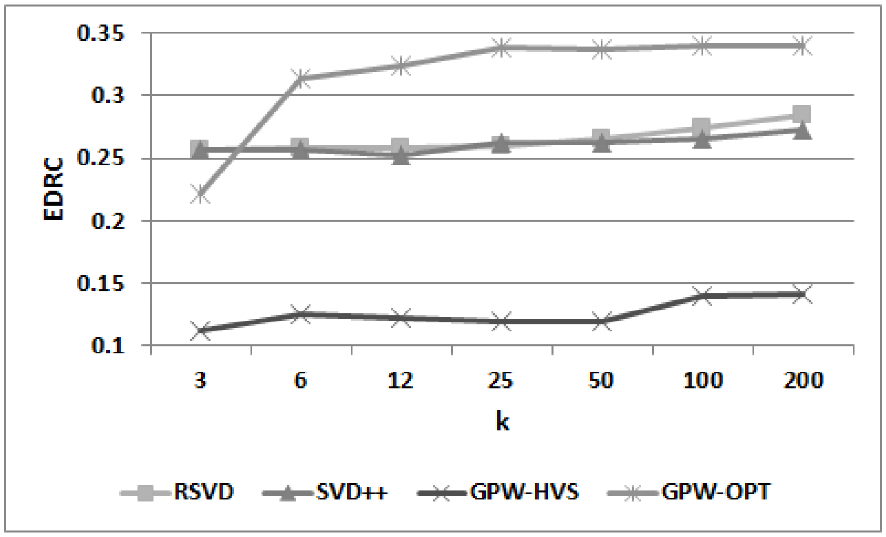


Figure 6.6: EDRC Comparison for Pairwise Preferences on MovieLens 100K

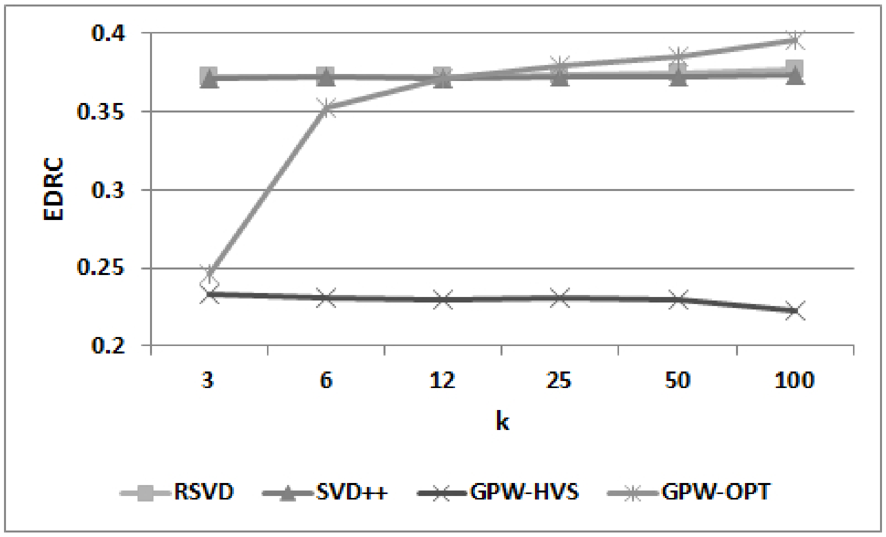


Figure 6.7: EDRC Comparison for Pairwise Preferences on MovieLens 1M

the result of RSVD or SVD++ and holds a small lead of again around .0015 up to .006 for k=200.

Figures 6.6, and 6.7 show the results comparing against related work for EDRC. Similar to pairwise loss and nDCG, the same relative order for results occurred in all instances. For EDRC, the maximum value is 1 and the lowest value is -1. For the MovieLens 1M figure, I again used a different

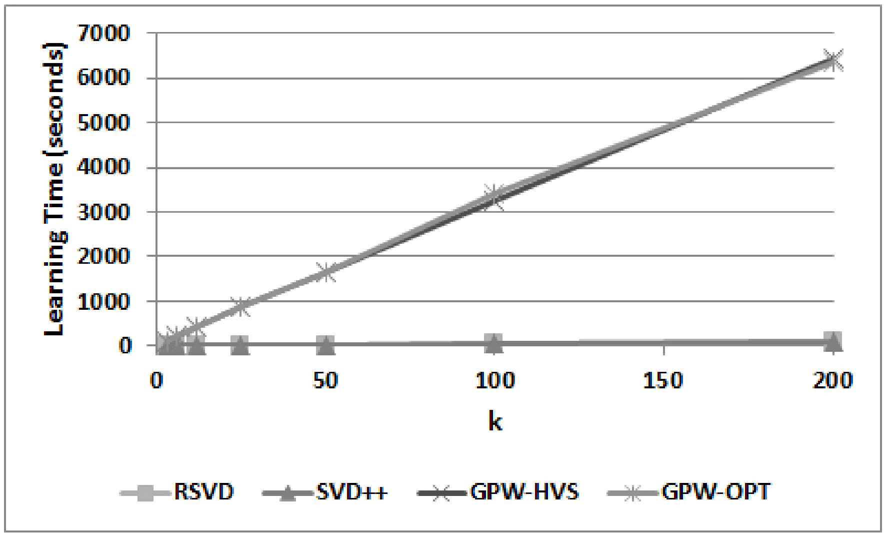


Figure 6.8: Learning Time Comparison for Pairwise Preferences on MovieLens 100K

learning and regularization rate in order to beat RSVD and SVD++. For the MovieLens 1M figure, the learning rate was .00075 and regularization rate was .014. The combination allowed me to beat the related work. For MovieLens 100K, my method takes until k=6 to beat RSVD or SVD++ and holds a steady lead of .05 to .065 all the way to k=200. For MovieLens 1M, it take until k=25 to beat RSVD or SVD++ with a margin of around .01 increasing up to .02 for k=200.

Figures 6.8 and 6.9 show the results for the learning times of the different approaches. Learning time is only the time it takes to learn the parameters of the model. Learning time is measured in seconds. The same labels are used on the table and the graph for the different methods. Notice that RSVD has a very negligible learning time on this dataset. Both of my methods have a learning time of around 2 hours for MovieLens 100K with 200 latent features. For MovieLens 1M with 100 latent features, this increases up to 17 hours. This is because although the increase of ratings is only 10 fold, the

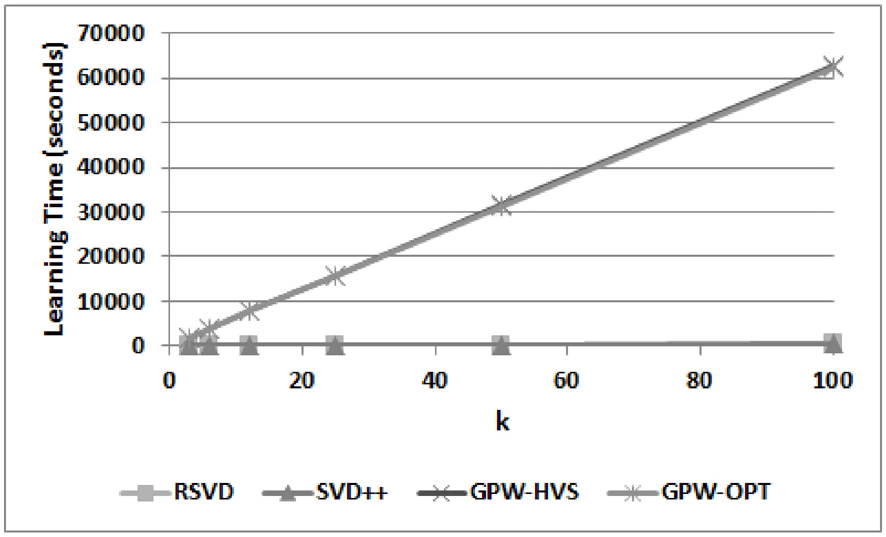


Figure 6.9: Learning Time Comparison for Pairwise Preferences on MovieLens 1M

number of preferences generate is much greater. Although this time seems to be very long, this is only associated with learning the parameters of the model. Prediction for a single user on a small set of items can occur almost instantly as prediction can happen in constant time. Knowing this, the time it takes to learn the model is not as important as the quality of result.

It is also worth noting that some optimizations to implementation needed to occur in order to achieve linear increase in time for increasing the number of latent features, k. Since there are many times where a prediction value needs to be calculated to determine loss, caching of predictions was very important to save time. For each set of 15 passes, one feature is being updated. I would cache the value of a prediction up to the current feature for all of the dot products of user and item vectors. Note that I only cache these dot products as other things cannot easily be cached such as multiplying the dot product of two vectors. I would cache the dot products and would perform the multiplication each time. I did not cache with the multiplication because when

Table 6.1: Prediction Time for MovieLens 100K in Seconds

	RSVD	SVD++	GPW-HVS	GPW-OPT
k=3	0.1248002	0.1404002	2.4172043	4.9608086
k=6	0.1404	0.1092001	2.6020042	2.4164043
k=12	0.1716002	0.1872003	2.6800037	2.580804
k=25	0.2340003	0.2652004	2.7980045	2.4500034
k=50	0.4056007	0.4212006	2.9796053	2.9484052
k=100	0.639601	0.7488015	3.74400669	3.7120061
k=200	1.3572025	1.3884025	3.7440066	4.8204084

Table 6.2: Prediction Time for MovieLens 1M in Seconds

	RSVD	SVD++	GPW-HVS	GPW-OPT
k=3	0.9204017	1.092002	49.9200877	41.527273
k=6	1.0452018	1.107602	49.6284805	41.5584728
k=12	1.1700022	1.27920233	50.9652896	44.9428757
k=25	1.4976027	1.6068027	49.7796875	45.3346797
k=50	2.1372037	2.1996038	53.7264943	50.7836853
k=100	3.6192064	3.8532066	65.2861146	60.1849057

the the scalar factor which is being multiplied changed, you would have to recalcuate the dot product in order to have the correct prediction. When calculating the prediction, I would use the cached value, then calculate the value for the current feature, and then I could carry out the calculation by adding the dot product for the additional features in one line because of my knowledge of the initial value of each feature. This would save the extra loops that are unnecessary as for all features beyond the feature currently being learned.

Prediction times are reported in Tables 6.1 and 6.2. The reported predictions times are the total for predicting every rating or preference in each of the 5 folds. Looking at both SVD-based methods, there is not a statistical difference between predictions times for RSVD and SVD++. Predictions increase with the increase in number of latent features and for k=200, the average is around .01 milliseconds. The difference in prediction time between ratings and preferences is due to system overhead such as loading data into memory or setting up data structures. For MovieLens 1M, there still is an average prediction time for each preference to be far less than 1 millisecond. This shows that although the learning time for my model is rather large, prediction can be done online with a delay unnnoticeable by the user.

Prediction for RSVD and SVD++ can be done by taking the dot product of the item and user vectors. This would require k number of multiplications and $k-1$ addition operations which can be done in under 1 millisecond for reasonable values of k. Given a set of items to rank for my method, the benefit of each item may be precomputed with the same number of operations. In order to make pairwise comparisons, these precomputed values can be used with one additional multiplication operation for α_u and one additional addition operation to find the value of one item given the other item. The total number of these operations depends on the number of preferences needed to be predicted, but only required two extra arithmetic operations per preference. Overall, these can still be completed in under 1 millisecond which is negligible to the user.

Figure 6.10 shows the results for varying the number of pairwise preference with respect to the total number of preferences generated for the MovieLens 100K dataset. Experiments were run for learning time on 20% 40%, 60%, 80%, and the entire set of pairwise preferences. It is evident that the system linearly scales with the number of pairwise preferences while holding k constant.

Next, I look at varying the parameters of the model to see what effect it has on quality. Figures 6.11 and 6.12 look at varying the learning rate and its effect on loss and nDCG. Figures 6.13 and 6.14 look at varying the

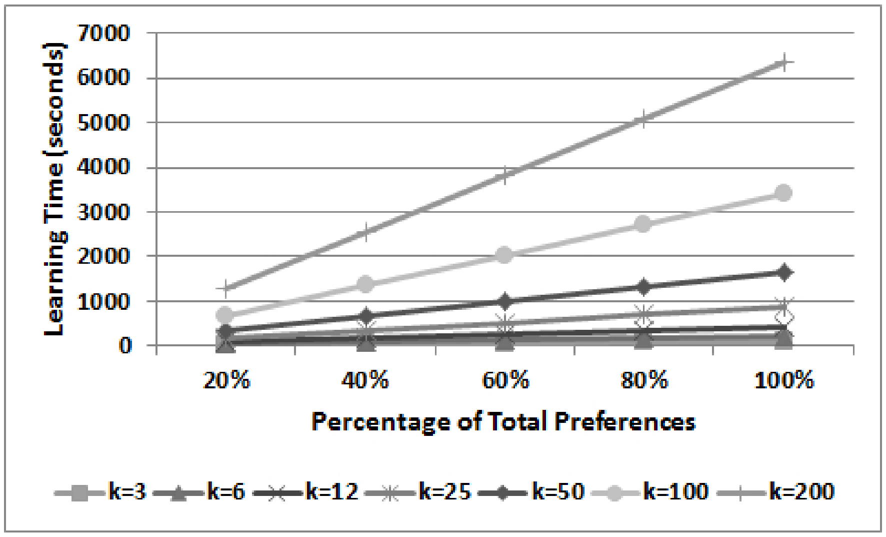


Figure 6.10: Learning Comparison for Varying Number of Pairwise Preferences

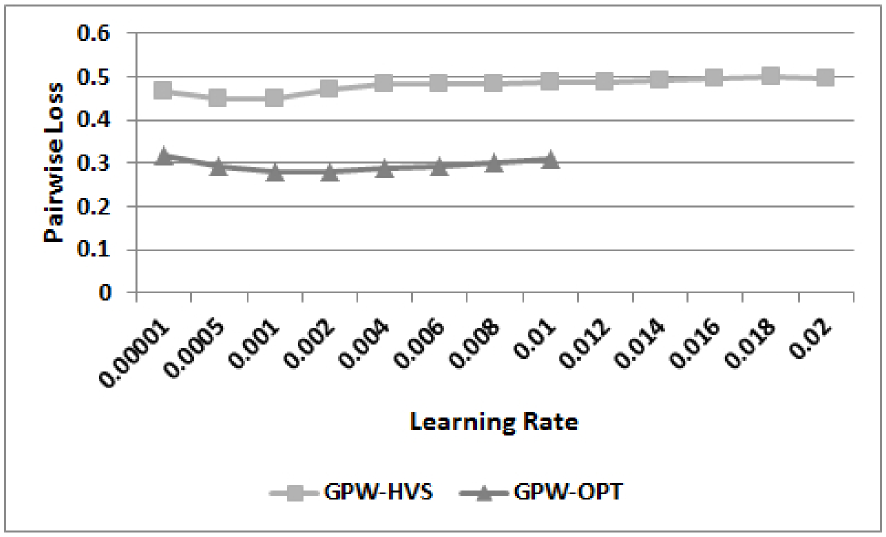


Figure 6.11: Pairwise Loss Comparison of Changing Learning Rate for Pairwise Preferences

regularization rate and its effect on loss and nDCG. For results on pairwise loss, the range for learning rates was .0001, .005, .001, .002 to .02 with a step of .002 for this span and regularization rates was from was .001 and .002 to .024 with a step of .002 for this span. The number of latent features was 50. For results on nDCG, the span of learning rates was .0005, .00075, .001, and

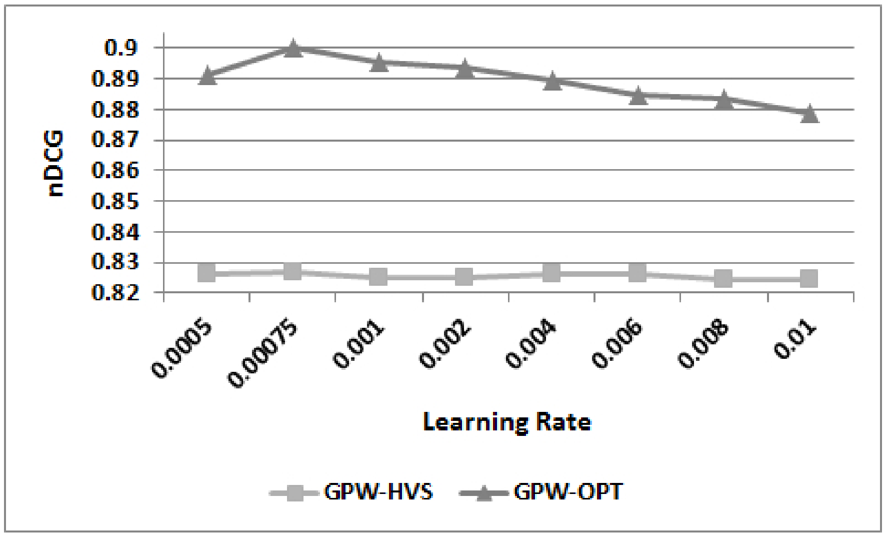


Figure 6.12: nDCG Comparison of Changing Learning Rate for Pairwise Preferences

.002 to .01 with a step of .002 and for regularization rates the range was .002 to .02 with a step of .002. The number of latent features was 200. All results are shown based on the MovieLens 100K dataset.

Figures 6.11 and 6.12 look at the learning rate. First, please note there are missing data points for GPW-OPT from a learning rate of .012 to .02. These learning rates were too large and cause the system to behave unpredicately yielding features whose values who went outside the range of possible values for the data type used for features. To save on RAM, I used a 32-bit floating point number to store features. Although a float has a rather large range of possible value, when the system consistently overcorrects, pass over pass, the values for a float can quickly be very large or small. This occurs when the learning rate is so large that it updates the model too far in one direction. The next time this is updated, it updates it too far in the other direction. This will continue until the updating goes beyond the range of possible values for its datatype. This can be fixed by increasing the reguarlization rate to a larger value. These values are omitted for that reason. For pairwise loss, I use a regularization rate of .02 and for nDCG, I used a regularization rate rate of .014. For both graphs I see that the best results occur at or near where I presented results in the previous figures, learning rate of .002 and regularization rate of .02. For GPW-HVS, the best results occur at different places in the range of learning rates and have increasing and descreasing results moving within this span. Heaviside loss proves to be instable by not producing smooth results as seen in GPW-OPT. The drawbacks of Heaviside loss are discussed in Section 3.6. The error calculation in Line 4 of Algorithm 1 will always be either 0 or 1 for Heaviside loss whereas for GPW-OPT, it will be the difference between the current prediction and the optimization goal. Heaviside loss may be 1 when the difference between the current and desired prediction is very small (e.g. .001). This will change the model more drastically that it needs to and cause unexpected results for varying values for learning rate since each passes does not try to make the prediction value be any specific value like GPW-OPT. Instead, it just cares that one value is greater than another value. See Section 3.6 for further discussion on this topic. Looking at the results for nDCG, the best results occurs at a learning rate of .00075 with results sloping away from this point with the change in learning rate. Results seem to decrease more quickly as the learning rates increase closer to .01.

Figures 6.13 and 6.14 look at the regularization weight. For pairwise loss, I use a learning rate of .002 and for nDCG a learning rate of .0075. Again, for both graphs I see that the best results occur at where I presented results in the previous figures. For loss the best regularization rate is .02. For nDCG, the results are best at .014 and slope away from this point. Again, the same trend

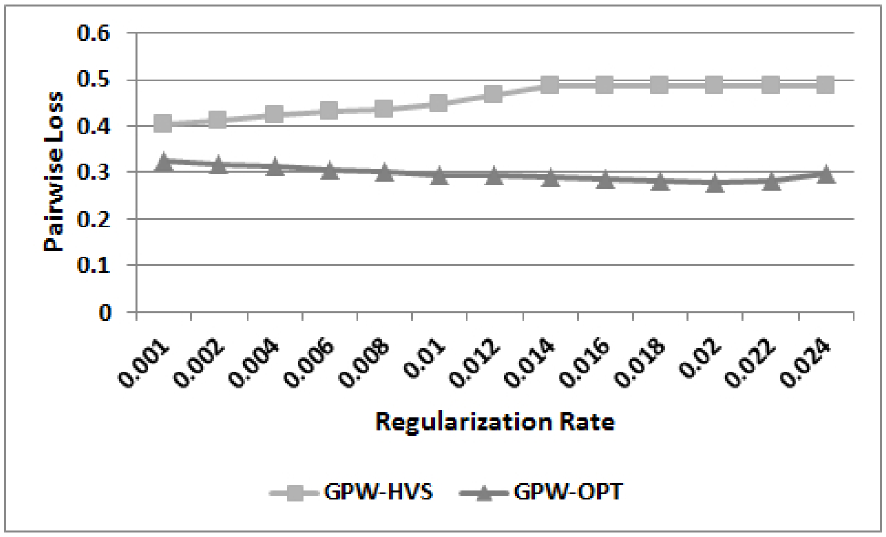


Figure 6.13: Pairwise Loss Comparison of Changing Regularization Rate for Pairwise Preferences

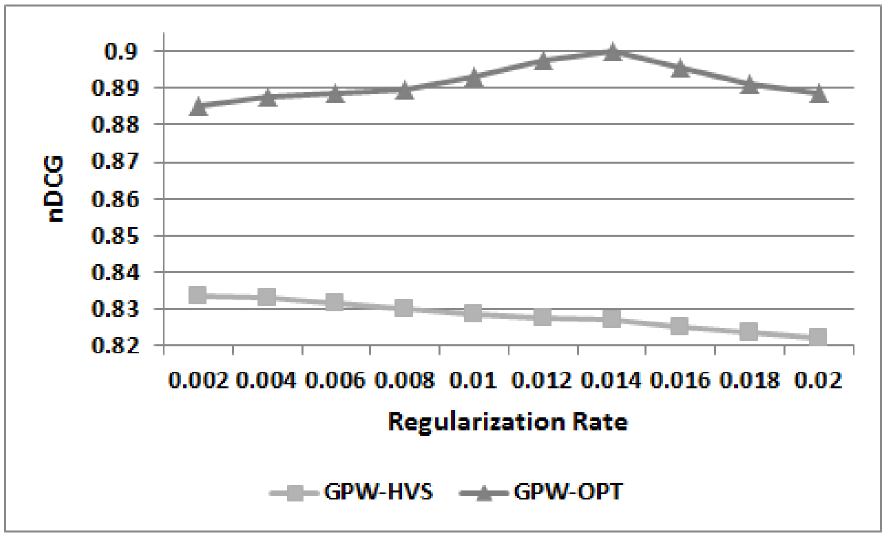


Figure 6.14: nDCG Comparison of Changing Regularization Rate for Pairwise Preferences

of results sloping away faster as the regularization rate more farther from .014 is also seen in this figure.

Modified Pairwise Preference Problem Setting Results

For this section, I modifed the way preference folds are generated to be closer to the problem setting I am trying to work on. In order to do this, I generate rating folds seperately from preference folds. Nothing changes to the way that I generate rating folds and I use the exact same random rating folds for the experiments of this section. The difference is in how I generate preference folds. To generate the new preference folds, I first generate all preferences based on the original dataset such that all ratings with a higher value are preferred to those of a lower value. For example, for a given user, all items rated 5 are preferred to those rated 4, 3, 2, and 1. After these preferences are generated, I then randomly move them into each fold. The difference of having folds of preferences versus folds of ratings, like RSVD and SVD++, is done as there is a difference in problem setting between my methods and that of the SVD-based methods. The figures of this section reflect the different way of generating preference folds.

Again, I first look at the pairwise preference, or global, problem setting as seen in Chapter 3. I ran experiments to compare with related work with respect to loss, nDCG, EDRC, learning time, and prediction time on both MovieLens datasets. Additionally, I show a comparison of changing learning parameters for the learning rate and regularization rate for my model which shows these changes for loss and nDCG. The work I compare against is RSVD [19] and SVD++ [33]. For both loss and nDCG, I look at comparing varying lengths of latent feature vectors which is denoted as k. I use the best parameters for my model which is a learning rate of .001 and regularization rate of .004. For the length of the latent feature vectors, I use 3, 6, 12, 25, 50, 100, and 200. Note, I truncate results for MovieLens 1M at 100 latent features due to the large learning time. Additionally, I use 15 training passes which is cited in literature, [19], to be enough for learning a model using stochastic

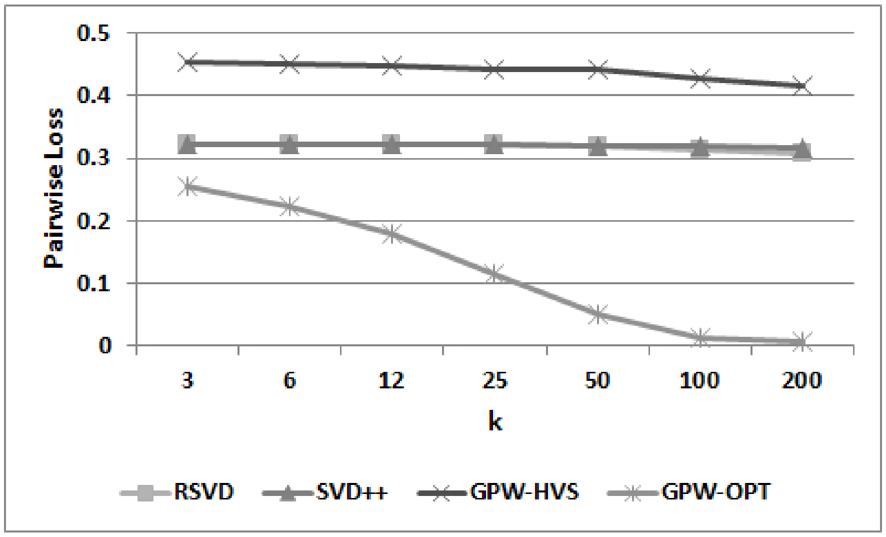


Figure 6.15: Pairwise Loss Comparison for Pairwise Preferences on MovieLens 100K

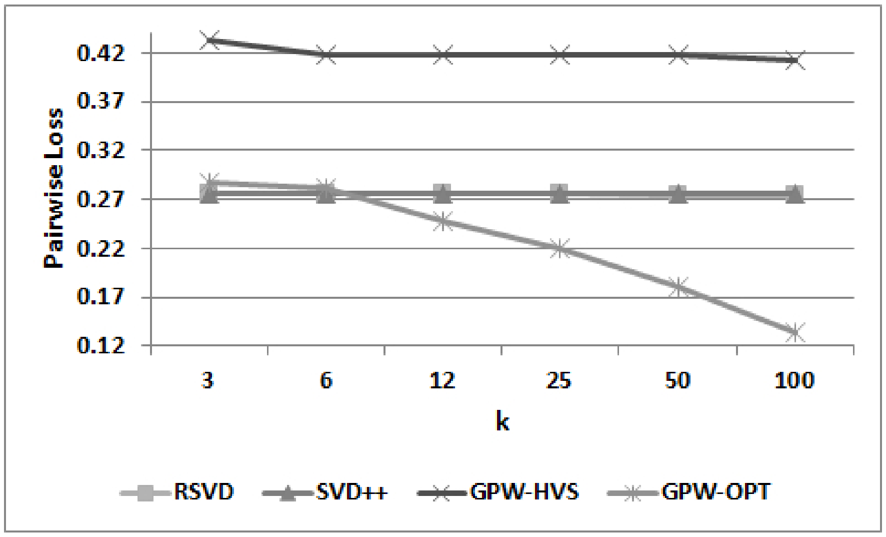


Figure 6.16: Pairwise Loss Comparison for Pairwise Preferences on MovieLens 1M

gradient descent. For RSVD and SVD++, I used a learning rate of .001, a regularization rate of .02, and 15 passes per features as used in [19].

Figures 6.15, and 6.16 show the results comparing against related work for loss. I used the label of GPW for my global pairwise model with HVS used for the Heaviside loss technique and OPT for numerical optimization. Looking at the Figure 6.15, which is for evaluation on loss on the smaller dataset, I see that all methods improve slightly when increasing the number of latent features, k, except for GPW-OPT which increases more rapidly with the increase in features. It is apparent that the Heaviside loss technique performs the worst. RSVD also performs significantly worse than my model with a numerical optimization technique. RSVD also doesn't appear to rank items better by increasing the number of latent features which is attributed to RSVD being for prediction accuracy. This is consistent with literature providing results on pairwise loss for SVD-based methods. In [35], the provide results for loss on SVD++ in Table 2 for 50, 100, and 200 features. From 50 to 100 features, loss improves .2 and from 100 to 200 features, loss improves .0014. Note that results are provided for FCP (frequency of concordant pairs) which is another form of pairwise loss, where 1 is the best result and 0 is the worst. Looking at my results for SVD++, the change for 50 to 100 features is .0017 loss and for 100 to 200 features, it is .0012 loss. This values are consistent with [35] with a better results increasing the number of features from 50 to 100 is not quite as good as 100 to 200. Looking at the larger dataset, there is a decrease in quality of the result for the same number of latent features. The difference is only .033 loss when k=3, but the difference is around .12 when k=100. There difference between these two figures is the dataset. Figure 6.15 is for MovieLens 100K and Figure 6.16 is for MovieLens 1M. This two datasets are disjoint in their sets of users. However, this does not explain why each of the other methods increase consistently between the two datsets by .03 to .05. The most probable explanation for this discrepancy is that I tuned my parameters for MovieLens 100K and used the exact same parameters for MovieLens 1M experiments. Since there are 10 times as many ratings in MovieLens 1M, there are at least that many more preferences as well. This large increase in preferences may require a larger regularization to prevent overfitting and/or a smaller learning rate to update the model less quickly because of the larger increase in training data points. Additionally, with both datasets, it can be seen that with smaller values of k, most of the methods converge to similar pairwise loss values. This is also reasonable because with fewer latent features, there is less ability for any model to capture users' preferences.

It is worth noting that in Figure 6.15, pairwise loss for GPW-OPT approaches 0 loss when k increases past 100 up to 200. This is likely attributed to a change of problem setting from item ratings to pairwise preferences amongst items. The folds for SVD-based methods consists of user, item, and rating tuples while for my problem setting they consists of tuples with a user and a preference comparing two items. For the SVD-based methods, for each user the sets of items in each fold are disjoint while in my problem setting, items may show up in many, if not all, of the folds. This allows for my model to better learn the features of an item before it has to prediction how a user will like the item in a different situation. For SVD-based methods, the only thing stored is a user rating. Since there is not previous information tell the model how well a user likes an item, predictions for these methods cannot produce as good of result. This likely explains why GPT-OPT can achieve almost no loss when k is greater than 100.

Figures 6.17, and 6.18 show the results comparing against related work for nDCG. I again use the same labels as pairwise loss for the various methods. Again, the same relative order is seen in the results with numerical optimization yielding the best results with RSVD and SVD++ being right behind and Heaviside loss being the worst. Please note that a higher value for nDCG indicates a better quality results with 1 being the best and 0 being the worst.

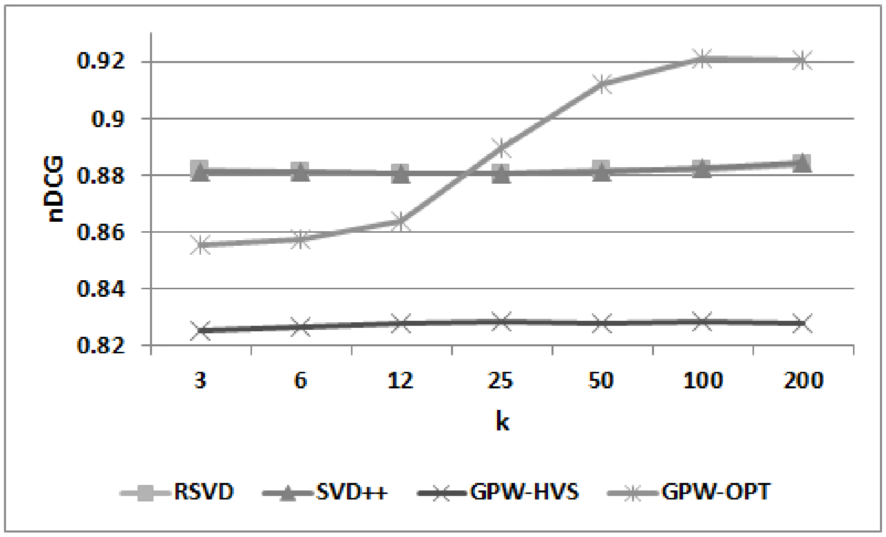


Figure 6.17: nDCG Comparison for Pairwise Preferences on MovieLens 100K

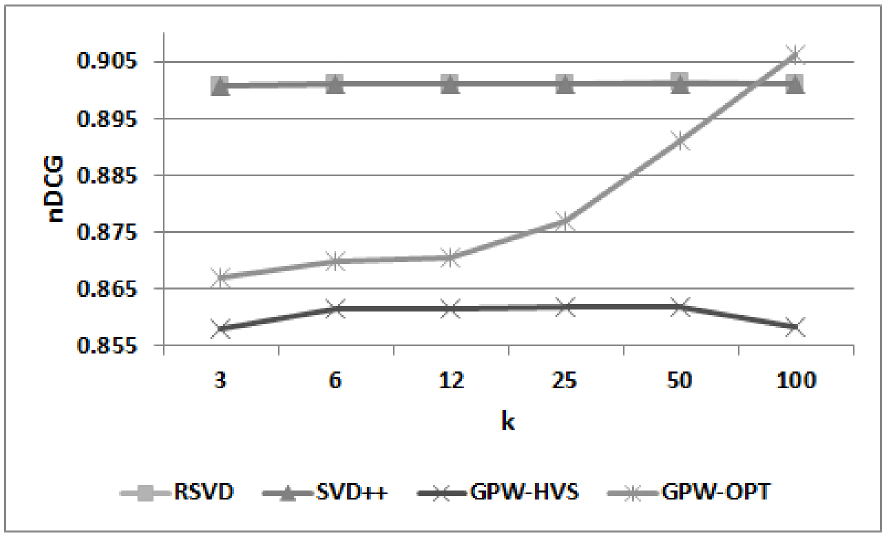


Figure 6.18: nDCG Comparison for Pairwise Preferences on MovieLens 1M

Better results are also seen with increasing the number of latent features, k. Similar trends show with slight increases with increasing the number of latent features with the numerical optimization increase more rapidly with increasing number of latent features.

It is worth nothing that SVD-based methods have better values for nDCG up to k=12 for Figure 6.17, and and k=100 for Figure 6.18. This is

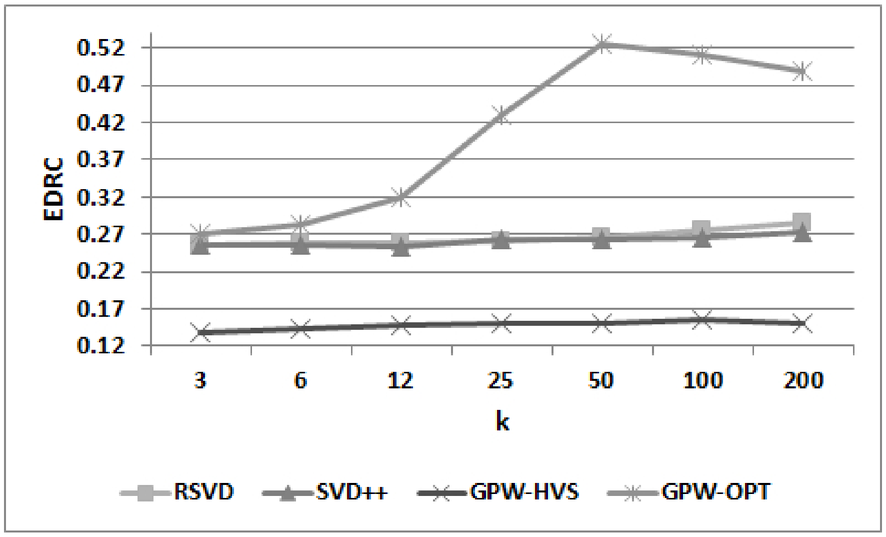


Figure 6.19: EDRC Comparison for Pairwise Preferences on MovieLens 100K

because of the difference of problem setting for the two methods as discussed why pairwise loss can almost reach 0 for GPT-OPT after 100 latent features. For evaluation of nDCG, I attempted to rank the items involved in the fold being withheld. For my problem setting, I need to rank those items based on the predicted preferences for that fold. Since it is possible that there was not preferences relating each possible pairwise combination of items in the fold, items may not be able to be ranked as well in comparison to a pointwise technique such as the SVD-based methods. The explains why it takes longer for nDCG to improve for my pairwise models.

Figures 6.19, and 6.20 show the results comparing against related work for EDRC. Similar to pairwise loss and nDCG, the same relative order for results occurred in all instances. Similar to nDCG, results with a higher value indicate a higher quality method. However, with EDRC, the maximum value is 1 and the lowest value is -1. With EDRC, similar trends occur with my method with numerical optimization growing more rapidly. There also seems to be a slight decrease in results for EDRC for this method when moving from 50 to

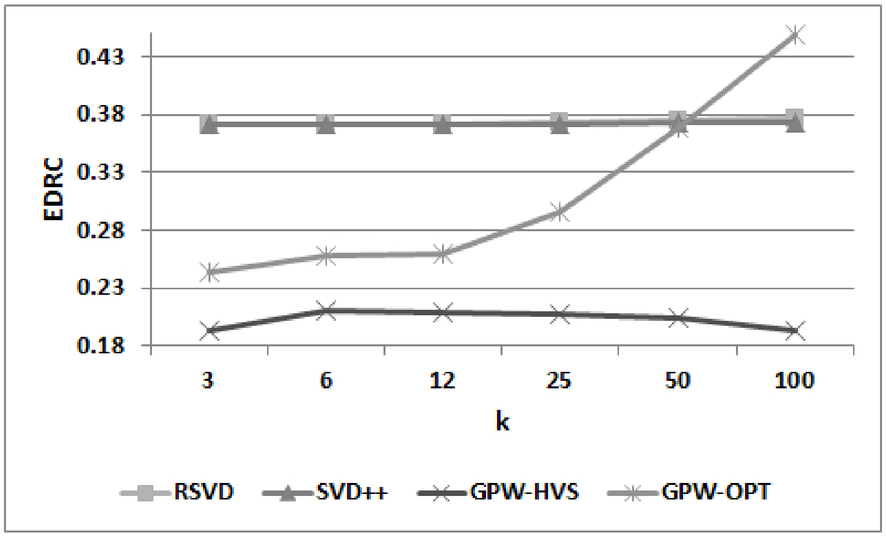


Figure 6.20: EDRC Comparison for Pairwise Preferences on MovieLens 1M

200 latent features. For nDCG, there seems to be a minimal decrease in nDCG from 100 to 200 latent features, but this is more noticeable for EDRC. Both nDCG and EDRC should follow similar trends as they are both weighted measures that discount for mistakes made while ranking based on the relevance of the item where the most preferred items are more heavily weighted. EDRC has a larger decrease than nDCG from k=50 to k=200 as nDCG uses logarithmic discounting as seen in Equation 2.7 and EDRC uses a linear discount. Additionally, EDRC discount items that have the same relevance (e.g. rated the same) using the same discount term whereas nDCG discount each item based on their index in a ranked list. When ranking n items, nDCG will discount the first item with $log_2(1+1)$=1 and the n^{th} item with $log_2(n+1)$. For EDRC when using a rating scale of 1 to 5, the largest discount term is 5. Looking at the results would show that the errors (e.g. pairwise inversions) for k=50 to k=200 are coming with more relevant items (e.g those rated 5) as with EDRC for all items rated 5, the discount would be 1 and for nDCG there would start at 1 and grow. Since the discount term is what is being

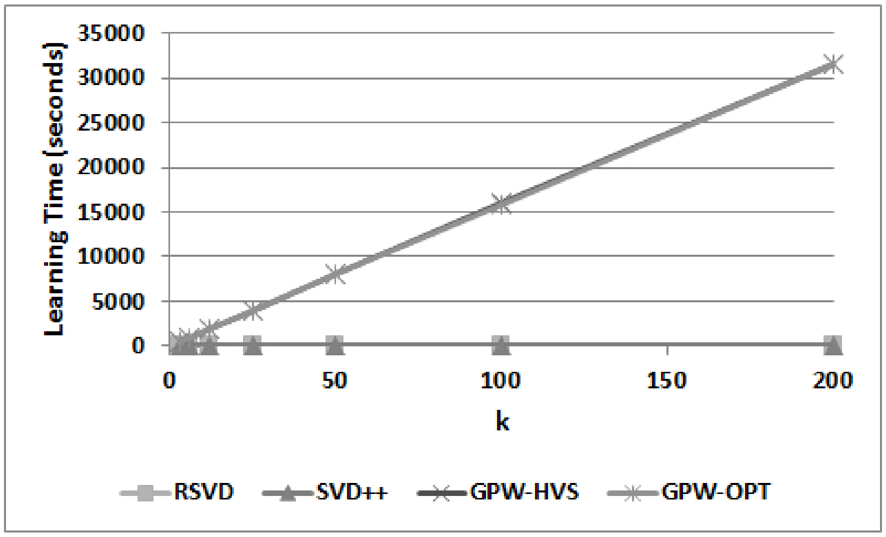


Figure 6.21: Learning Time Comparison for Pairwise Preferences on MovieLens 100K

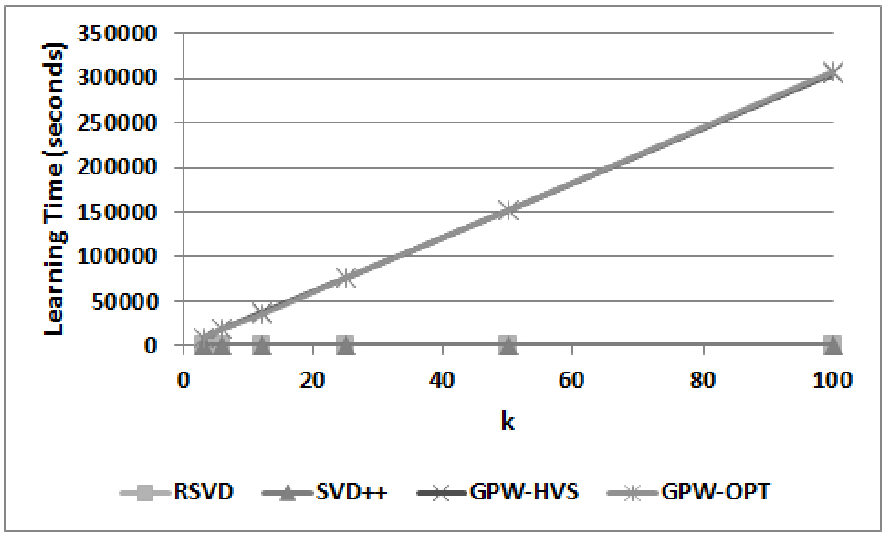


Figure 6.22: Learning Time Comparison for Pairwise Preferences on MovieLens 1M

dividing, the larger the value would have less of an effect for the less relevant items and a larger effect for the most relevant items. Because of this, it appears that the errors are occuring at higher rated items which explains the larger decrease for EDRC than nDCG.

Table 6.3: Prediction Time for MovieLens 100K in Seconds

	RSVD	SVD++	GPW-HVS	GPW-OPT
k=3	0.1248002	0.1404002	10.9668194	9.9372173
k=6	0.1404	0.1092001	11.1696195	10.296018
k=12	0.1716002	0.1872003	11.4348201	10.6080186
k=25	0.2340003	0.2652004	11.8872209	11.1852195
k=50	0.4056007	0.4212006	13.2600234	12.4488219
k=100	0.639601	0.7488015	15.5220272	14.6016256
k=200	1.3572025	1.3884025	19.1256336	19.3752341

Table 6.4: Prediction Time for MovieLens 1M in Seconds

	RSVD	SVD++	GPW-HVS	GPW-OPT
k=3	0.9204017	1.092002	229.3360028	189.7899334
k=6	1.0452018	1.107602	219.242785	190.6791349
k=12	1.1700022	1.27920233	222.4573907	194.9379424
k=25	1.4976027	1.6068027	231.0988059	205.1091602
k=50	2.1372037	2.1996038	256.4332505	225.9819971
k=100	3.6192064	3.8532066	289.5053087	266.0272672

Figures 6.21 and 6.22 show the results for the learning times of the different approaches. Learning time is only the time it takes to learn the parameters of the model. Learning time is measured in seconds. The same labels are used on the table and the graph for the different methods. Notice that RSVD has a very negligible learning time on this dataset. Both of my methods have a learning time of around 9 hours for MovieLens 100K with 200 latent features. For MovieLens 1M with 100 latent features, this increases up to 3.5 days. This is because although the increase of ratings is only 10 fold, the number of preferences generate is much greater. Although this time seems to be very long, this is only associated with learning the parameters of the model. Prediction for a single user on a small set of items can occur almost instantly as prediction can happen in constant time. Knowing this, the time it takes to the learning the model is not as important at its quality of result. Again, I use the same caching optimizations as found in the previous section.

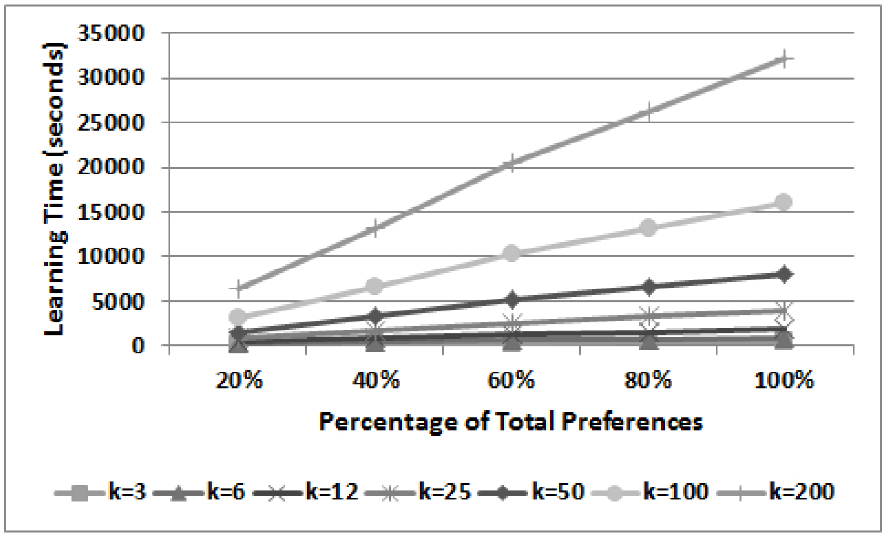


Figure 6.23: Learning Comparison for Varying Number of Pairwise Preferences

Prediction times are reported in Tables 6.3 and 6.4. The reported predictions times are the total for predicting every rating or preference in each of the 5 folds. Looking at both SVD-based methods, there is not a statistically difference between predictions times for RSVD and SVD++. Predictions increase with the increase in number of latent features and for k=200, the average is around .01 milliseconds. For my methods, there are around 7 million preferences generated from the 100,000 ratings. Predictions for each preference are magnitudes smaller than a millisecond. The difference in prediction time between ratings and preferences is due to system overhead such as loading data into memory or setting up data structures. For MovieLens 1M, there are around 13.7 million preference generate from 1 million ratings. This still yields the prediction time for each preference to be far less than 1 millisecond. This shows that although the learning time for my model is rather large, prediction can be done online with a delay unnoticeable by the user. Prediction times that are reported in the figures of this section, follow the same methodology spoken of in the previous section.

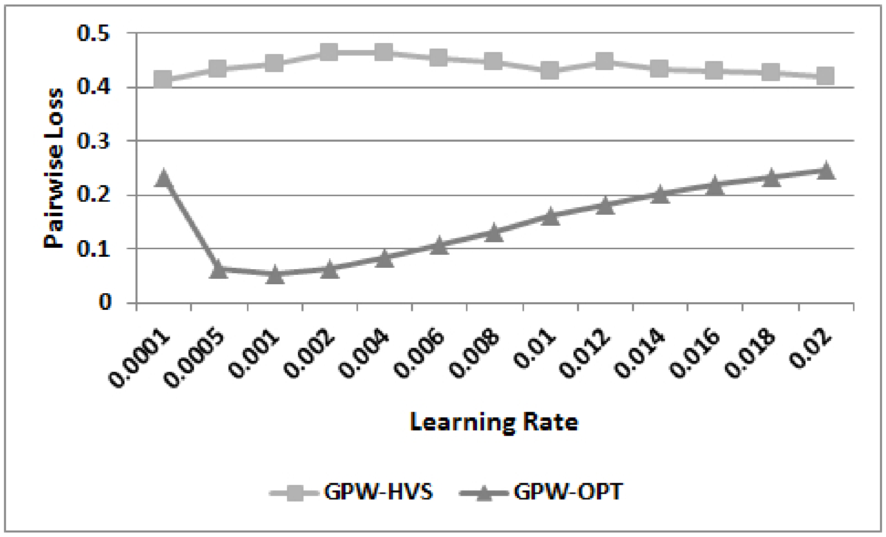


Figure 6.24: Pairwise Loss Comparison of Changing Learning Rate for Pairwise Preferences

Figure 6.23 shows the results for varying the number of pairwise preference with respect to the total number of preferences generated for the MovieLens 100K dataset. Experiments were run for learning time on 20% 40%, 60%, 80%, and the entire set of pairwise preferences. It is evident that the system linearly scales with the number of pairwise preferences while holding k constant.

Next, I look at varying the parameters of the model to see what effect it has on quality. Figures 6.24 and 6.25 look at varying the learning rate and its effect on loss and nDCG. Figures 6.26 and 6.27 look at varying the regularization rate and its effect on loss and nDCG. For all of these experiments, the range for both learning and regularization rates was from was .0001, .0005, .001 and .002 to .02 with a step of .002 for this span. For each of the figures looking at varying parameters, I used 50 latent features.

Figures 6.24 and 6.25 look at the learning rate. For both graphs I see that the best results occur at or near where I presented results in the previous

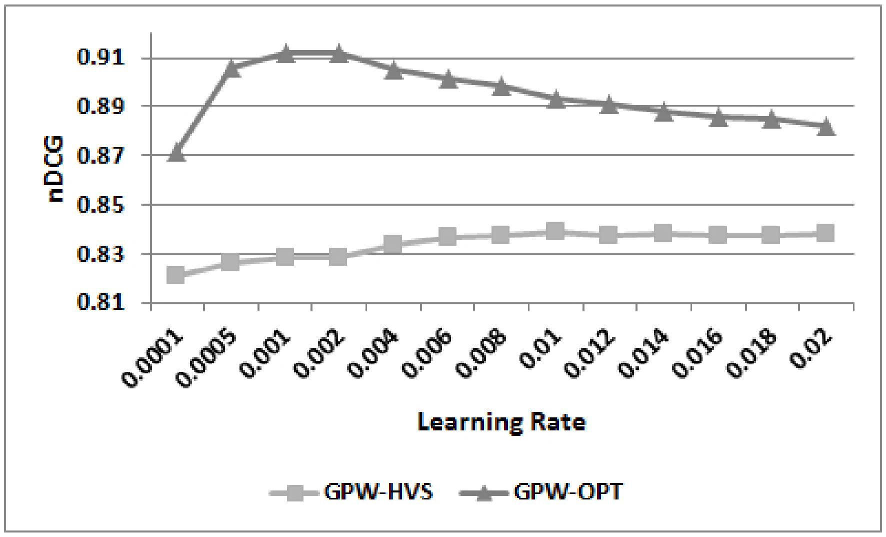


Figure 6.25: nDCG Comparison of Changing Learning Rate for Pairwise Preferences

figures, learning rate of .001 and regularization rate of .004. Results degrade moving away from these points at changing the learning rate affects the model. A learning rate that is too small does not update the model enough while large learning rates encourages overfitting. I can see that this is the case for my experimental results. The best results for the numeric optimization technique is found with a learning rate of .001 and it degrades by increasing the learning rate. GPW-OPT greatly improves with the jump from a learning rate of .0001 to .0005 and gets slightly better up to .001. It then slowly degrades moving up to a learning rate of .02. This same trend does not occur with GPW-HVS and the best results do not show up at the same values for learning rate. For GPW-HVS, the best results occur at the extremes of the range of learning rates (.0001 and .02) and have increasing and descreasing results moving within this span. Heaviside loss proves to be instable by not producing smooth results as seen in GPW-OPT. The drawbacks of Heaviside loss are discussed in Section 3.6. The error calculation in Line 4 of Algorithm 1 will always be either 0 or 1 for Heaviside loss whereas for GPW-OPT, it will be the difference

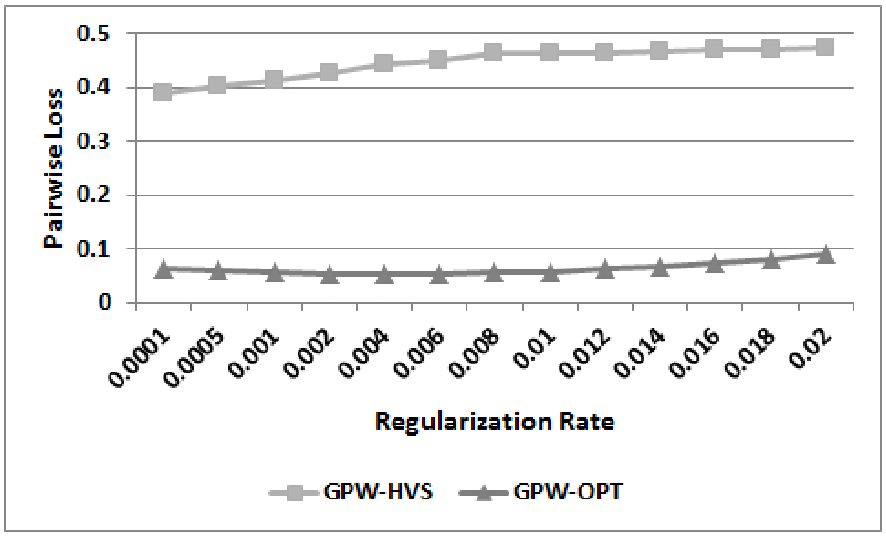


Figure 6.26: Pairwise Loss Comparison of Changing Regularization Rate for Pairwise Preferences

between the current prediction and the optimization goal. Heaviside loss may be 1 when the difference between the current and desired prediction is very small (e.g. .001). This will change the model more drastically that it needs to and cause unexpected results for varying values for learning rate since each passes does not try to make the prediction value be any specific value like GPW-OPT. Instead, it just cares that one value is greater than another value. See Section 3.6 for further discussion on this topic.

Figures 6.26 and 6.27 look at the regularization weight. Again, for both graphs I see that the best results occur at or near where I presented results in the previous figures. Small regularization rates allow for overfitting of the model, while large regularization weights may also hinder quality by underfitting the model. It is seen that a rate of .004 provides the best result while decreasing the rate allow for overfitting and degradation of quality occurs quickly. By increasing the rate, the loss of quality is not as prominent and occurs more gradually.

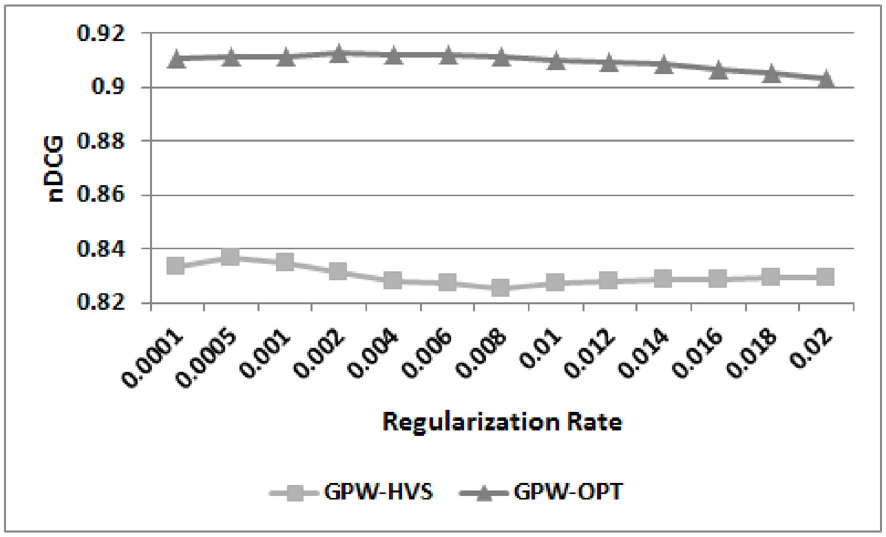


Figure 6.27: nDCG Comparison of Changing Regularization Rate for Pairwise Preferences

Session Problem Setting Results

Next, I look at evaluating the quality of my methods for the session problem settings seen in Chapter 4. I do this by comparing with relating work, CCF [75], and then evaluating my own methods.

I start by comparing with CCF to show that my methods outperform the only existing work that handles sessions. However, CCF has some limitations that require me to make changes to my problem setting in order to make a comparison. CCF only allows for one item to be the most preferred (e.g. purchased) and only allows for all items items to be in the less preferred relevance class. In order to allow for this, I must reinterpret the user study. To do this, I take the movie for each query that has the greatest x-position. In the case of having two or more movies with the same x-position value that are the most preferred, I randomly remove all but one of the movies . This allows for differing size offer sets. For example, is I remove one movie, the offer set is only 5 instead of 6. This is easily handled by taking the average benefit of the other items which is done in my model. This means that for our methods, there is always just two relevance classes with the most preferred class only having one item. Please note that I use Softmax loss for CCF as it was noted as being the best alternative for their method in their paper.

Additionally, I adopt other settings mentioned in the paper. These include discounting the learning rate by a factor of .9 after each feature is learned. Also, I use one of the best regularization rates mentioned in the paper which is .001. Please not that the best results in the paper are shown to be at a regularization rate of .0001, but my experiments at this value proved to be worse so I adopted a new value to allow CCF to be more competitive against my methods. Also, no learning rate was mentioned in the CCF paper, so I choose .01 after running some initial experiments. The combination of a learning rate of .01 and a regularization rate of .001 proved to yield the most competitive results. They also do not say how many training passes are made per feature. I used 15 since it is also cited in other literature.

For evaluation, I can use nDCG to compare with CCF as their problem setting only considers binary responses where user interactions can be interpreted as 1 and no interaction is 0. Our methods do not attempt to assign an explicit relevance value to an item and have varying numbers of ordinal relevance classes in each user session. This makes it difficult to evaluate using nDCG without making considerable alternatives to the nDCG evaluation procedure. However, because I am only comparing with CCF to begin with, I will use 1 for user interaction and 0 for no interaction to for nDCG.

For comparison with CCF, I use only use numeric optimization by relevance classes. This method outperforms the two other alternatives

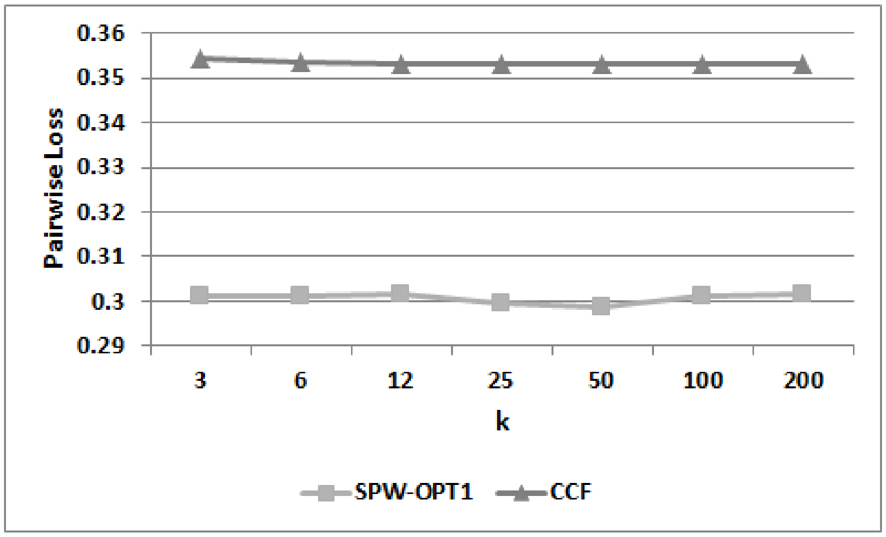


Figure 6.28: Pairwise Loss Comparison of SPW-OPT1 and CCF for Modified Session Problem Setting

(Heaviside loss and optimization by item/item pairs). I look at comparing varying lengths of latent feature vectors and I use the best parameters for my model which is a learning rate of .009 and regularization rate of .007. For the length of the latent feature vectors, I again use 3, 6, 12, 25, 50, 100, and 200. I also use 15 training passes.

Figure 6.28 has results for loss comparing with CCF. Looking at Figure 6.28, I can see that my model significantly outperforms CCF for all vector lengths. The difference in results between CCF and my model can be attributed the weaknesses of CCF such as using the traditional RSVD model for prediction which has context-independent relevance values. Results for SPW-OPT1 are very flat, never being more than .0016 away from .3 loss. Loss for CCF is also very flat going down to .3532 for k=12 to k=200. This flatness beyond k=12 is most likely because of discounting the learning rate by a factor of .9 after each feature is learned. Since the learning rate approaches 0 with increasing the number of features, the changes to model become smaller and smaller with each additional feature.

It is also worth noting that trends for the session methods appear much flatter than those of the previous figures for my pairwise preference models. There are two factors that contribute to this. The first is that for the session methods, there were no publicly available datasets that contained session data such as those found in the CCF paper. The user study that I conducted was on a small scale and only contained 100 users and 25 items. This is much smaller than other datasets as it only contained 3,000 data points with 100 users, with 6 movies per session, and 5 sessions per user. Because of only having 3% of the data compared to the smallest MovieLens dataset, there are less features that can distinguish such as small set of users and items. This explains why only need a limited number of features to achieve the best results. The second factor is that both the session pointwise model and CCF are pointwise models compared to the global pairwise which is pairwise. The pairwise model showed to be much better than the pointwise counterparts of RSVD and SVD++ in the previous figures which also has very flat trends. Since RSVD and SVD++, both pointwise, has flat trends, it makes sense why SPW-OPT1 and CCF did too. For more information regarding pairwise versus pointwise, please refer to Section 2.3.

Figure 6.29 have results for nDCG comparing with CCF. nDCG follows a similar trend to pairwise loss for both methods. Results for SPW-OPT1 are very flat never moving away from .675 more than .0025. CCF gets slightly better from k=3 to k=50 and then stays flat at .66.

Figure 6.30 has results for loss comparing with CCF. Again, trends are very flat for both methods. For SPW-OPT1, values never get more than .003

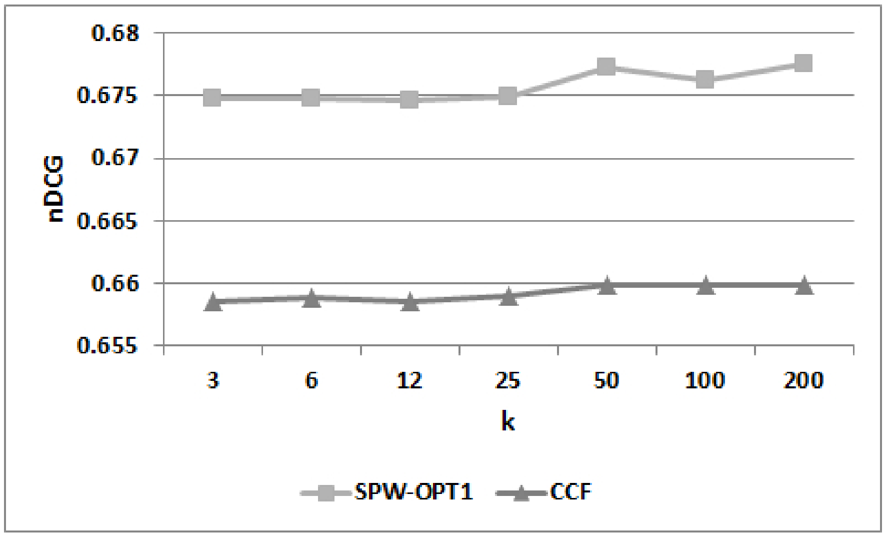


Figure 6.29: nDCG Comparison of SPW-OPT1 and CCF for Modified Session Problem Setting

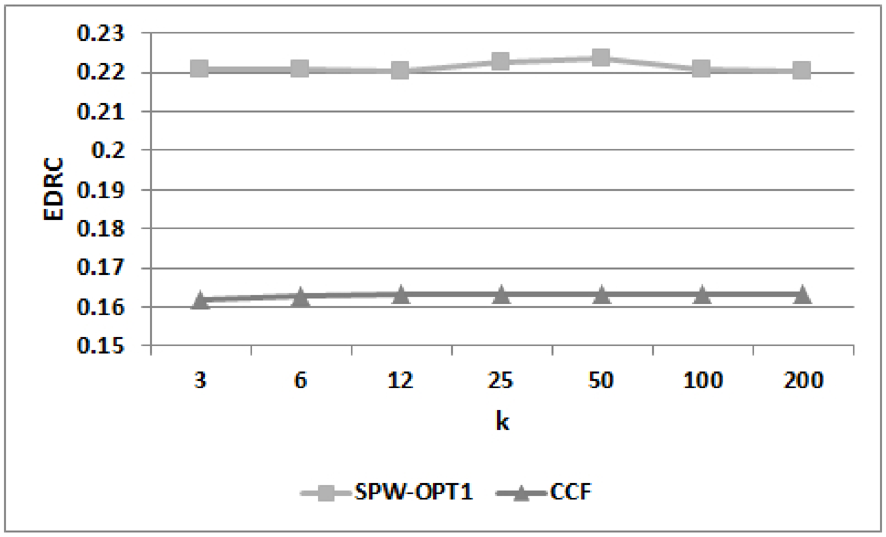


Figure 6.30: EDRC Comparison of SPW-OPT1 and CCF for Modified Session Problem Setting

from .22. For CCF, results increase up to k=12 where the value stays flat at .163.

Figure 6.31 show the results for the learning times for my model and CCF. Again, learning time is only the time it takes to learn the parameters of

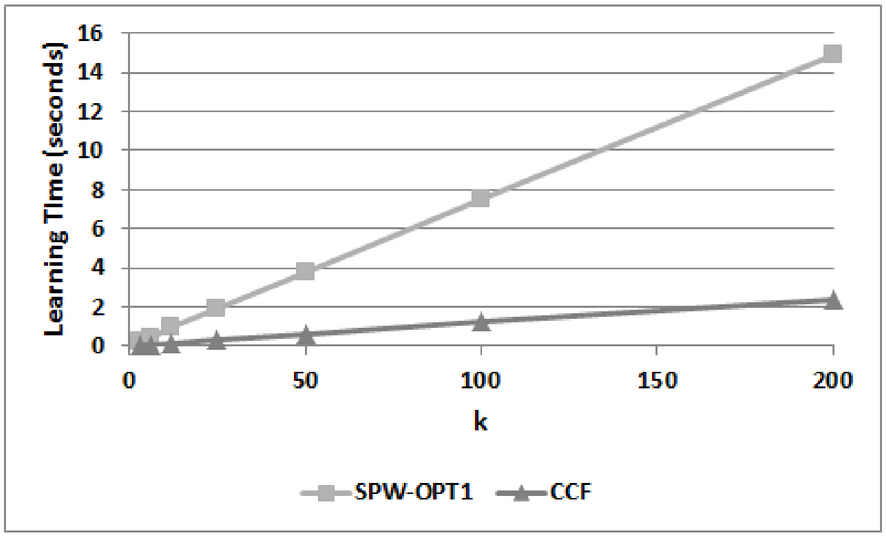


Figure 6.31: Learning Time Comparison of SPW-OPT1 and CCF for Modified Session Problem Setting

Table 6.5: Prediction Time for CCF and SPW-OPT1 in Seconds

	CCF	SPW-OPT1
k=3	0.0156	0.0156
k=6	0.0312001	0.0780001
k=12	0.0468001	0.0156001
k=25	0.0780001	0.0156
k=50	0.0780001	0.0312001
k=100	0.0936001	0.0468002
k=200	0.1716003	0.1092001

the model. Prediction time is the time it takes to predict the relevance of one item. Learning time is measured in seconds. Like RSVD, CCF has negligible a learning time compared to my method. This can be attributed to the simple update rules of CCF. However, I note that learning time for my learning methods are associated with offline computations which are not as important as they are not experienced by the user.

Prediction times for session methods are shown in Table 6.5. The general trend is that the prediction time increases with an increase in k, however this is not always the case. Predictions times for k and both methods are always less than .2. The variation for non-linearly increasing predictions times is most likely due to other system operations interrupting prediction time. However, with the total prediction time being at most .2, this means that one session can be predicted in at most .4 milliseconds. This value is very negligible and would not be perceived by the user.

Prediction time for all methods and all values of k are negligible as they are under 1 millisecond which was the smallest amount of time I was able to measure. This is same as the global problem setting. Prediction for CCF is the same as RSVD and SVD++ which can be done by taking the dot product of the item and user vectors. This would require k number of multiplications and $k - 1$ addition operations which can be done in under 1 millisecond for reasonable values of k. Given a set of items to rank for my method, I can precompute the benefit of each item with the same number of operations. In order to make a value prediction for each item, I can take these precomputed values and do $n - 2$ addition operations to get the sum for the opportunity cost where n is then number of items in the offer set. To finish the value prediction, one multiplication operation for α_u and one additional addition operation to find the total sum is needed. Overall, this can again be completed in under 1 millisecond which is negligible to the user.

Next, I look at evaluating the various alternatives for my opportunity cost model. I look at all three alternatives: Heaviside loss, numeric optimization by relevance classes and by item/item pairs. On all figures, Heaviside loss is denoted as HVS, numeric optimization by relevance classes by OPT1, and numeric optimization by item/item pairs by OPT2. I again compare at varying lengths of the latent features vectors and use the best parameters for my model which is a learning rate of .009 with regularization at .007. Again, I use

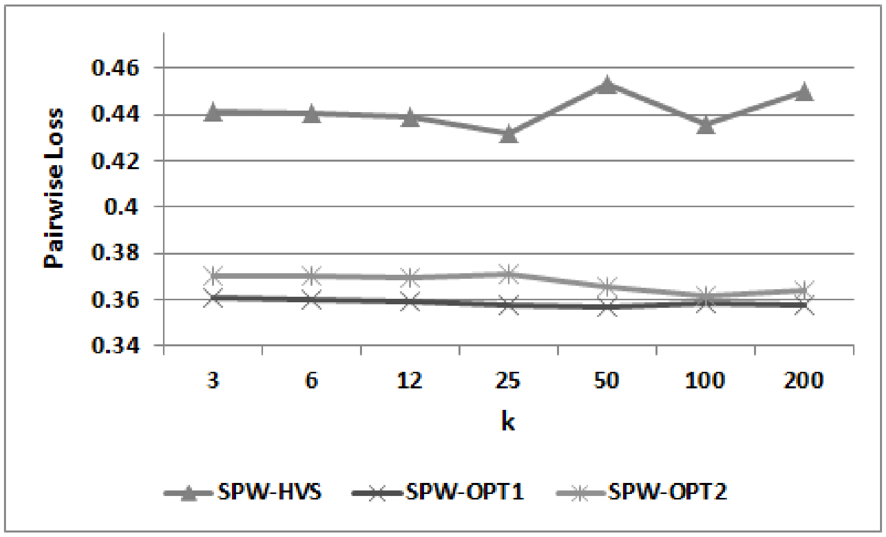


Figure 6.32: Pairwise Loss Comparison for Sessions

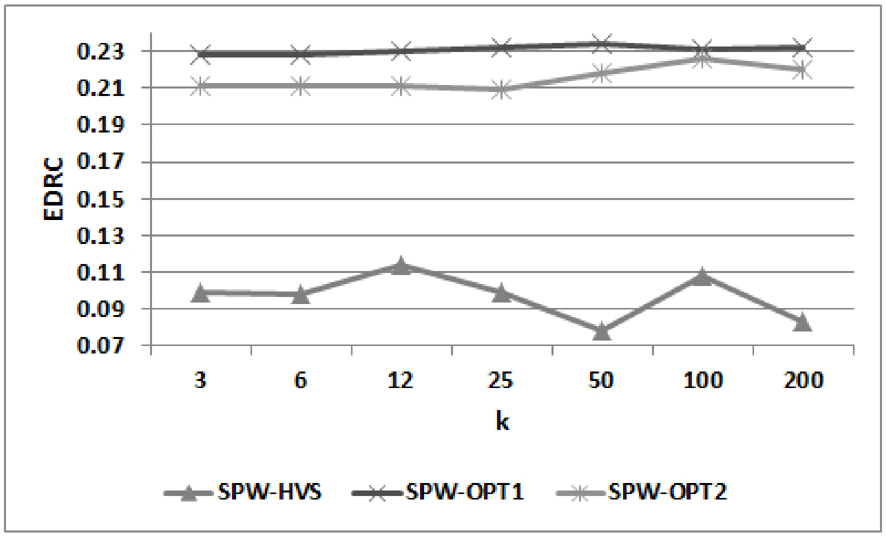


Figure 6.33: EDRC Comparison for Sessions

15 training passes. I evaluation on both loss and EDRC. I used EDRC instead of nDCG because nDCG requires explicit relevance values which are not found in this problem setting. EDRC shows its true value in this circumstance where nDCG is inapplicable due to limitations in which problem settings work and that explicit relevance is needed for its calculation.

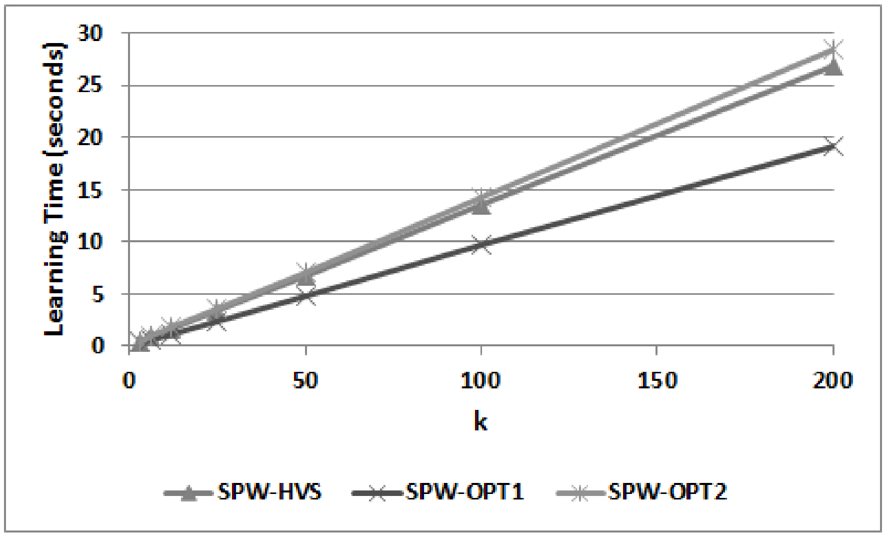


Figure 6.34: Learning Time Comparison for Sessions

Figures 6.32 and 6.33 show the results for loss and EDRC for my various techniques. Looking at Figure 6.32, it is seen that Heaviside loss is by far the worst alternative because of the way it assigns error. Out of the two remaining methods that both use numeric optimization, optimization by relevance classes (SPW-OPT1) outperforms optimization by item/item pairs (SPW-OPT2). This can be attributed to a reduction in the number of parameters of the model. Fewer parameters reduces model complexity and usually produces better results. Like previous figures, I see that trends are flat for SPW-OPT1 and SPW-OPT2 and that SPT-HVS is less stable. Both of these trends were described with the previous results for comparison with CCF. Note that EDRC is on a scale of -1 to 1 where 1 is the best and -1 is the worst.

Figures 6.34 show the results for learning times for my various techniques. Again, learning time is only the time to learn the model. Learning time is measured in seconds. I can see that all alternatives have almost identical running time. This is because they all gs through roughly the same process and the only different is the number of parameters to update which is

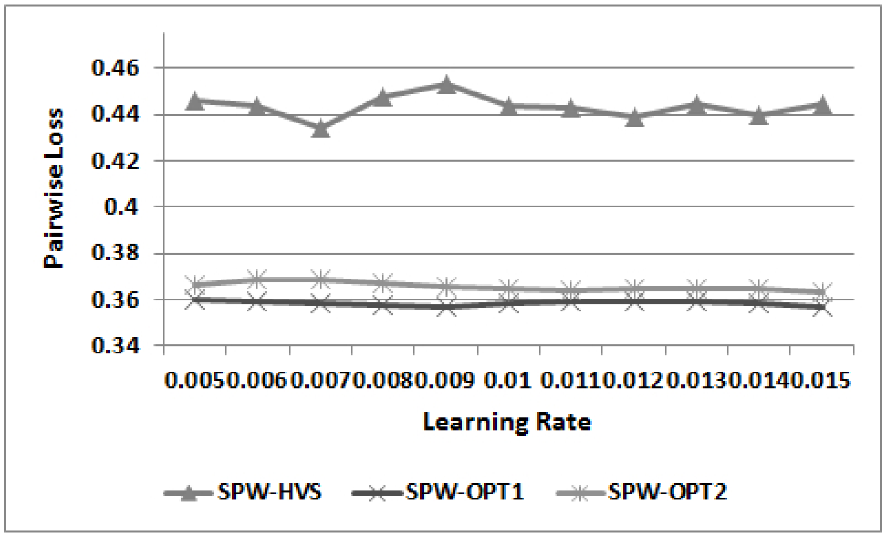


Figure 6.35: Pairwise Loss Comparison of Changing Learning Rate for Sessions

negligible with respect to all update actions. For 50 features, the learning time is roughly 5 seconds. For 100 features, the learning time is roughly 10 to 11 seconds and for 200 features, the learning time is roughly 20 to 25 seconds.

Prediction for this method is exactly the same as discussed in the previous analysis for the comparison with CCF.

Next, I again look at varying the parameters of the model to see what effect it has on quality. Figures 6.35 and 6.36 look at varying the learning rate and its effect on loss and EDRC. Figures 6.37 and 6.38 look at varying the regularization rate and its effect on loss and EDRC. For all of these experiments, the range of the learning rate and regularization is from .005 to .015. I look at different values for this set of experiments than previous ones because this dataset is much smaller and requires different parameters to find the best results. For each of the figures looking at varying parameters, I used 50 latent features.

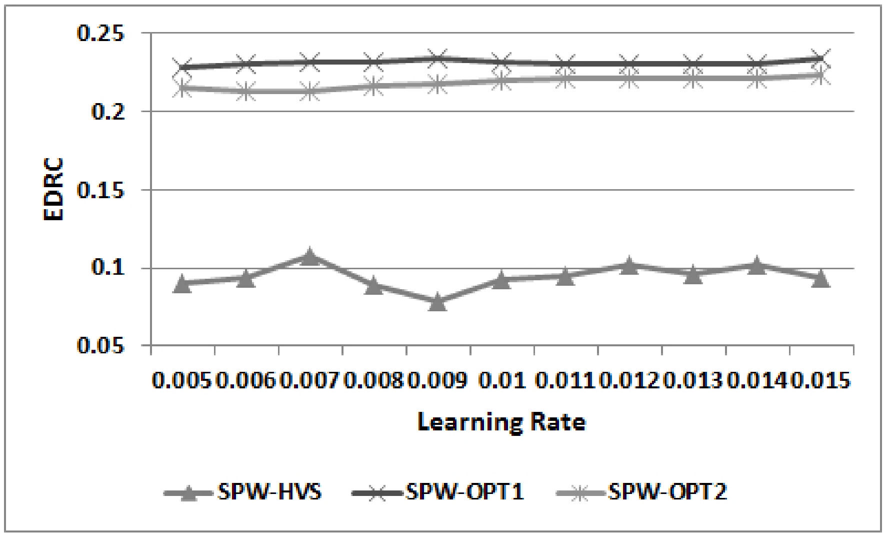


Figure 6.36: EDRC Comparison of Changing Learning Rate for Sessions

Figures 6.35 and 6.36 look at the learning rate. For both graphs I see that the best results occur at or near where I presented results in the previous figures, learning rate of .009 and regularization rate of .007. Results degrade moving away from these points at changing the learning rate affects the model. As mentioned before, small learning rates do not update the model sufficiently while large learning rates encourage overfitting. The best results for the numeric optimization by relevance classes (SPW-OPT1) technique is found with a learning rate of .009 and it quickly degrades by increasing the learning rate. This phenomena can be see for all methods while Heaviside loss is not as prominent. The trend for Heaviside loss is not stable as described previously because of its limitations on how it assigns error for the update rules.

Figures 6.37 and 6.38 look at the regularization rate. Again, it can be seen that that the best results occur at or near where I presented results in the previous figures, learning rate of .009 and regularization rate of .007. Results away from these points at not as good as small regularization rates allow for

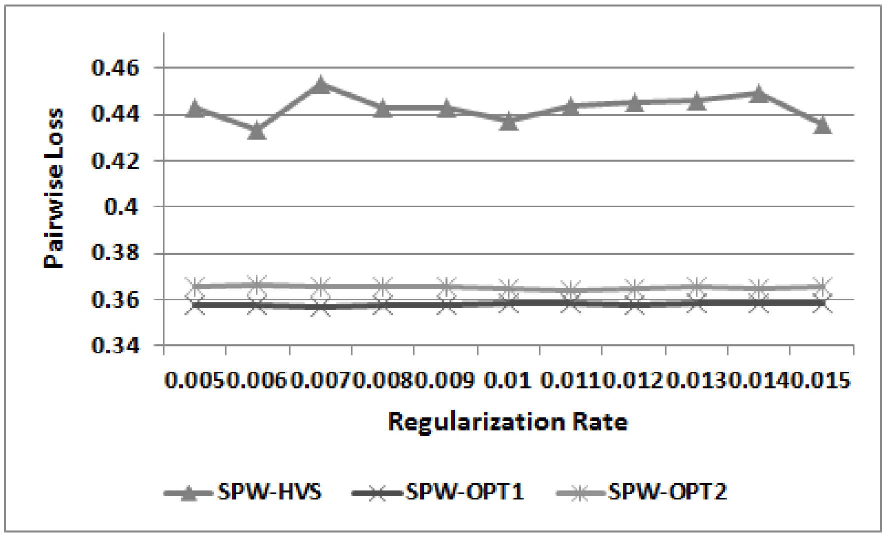


Figure 6.37: Pairwise Loss Comparison of Changing Regularization Rate for Sessions

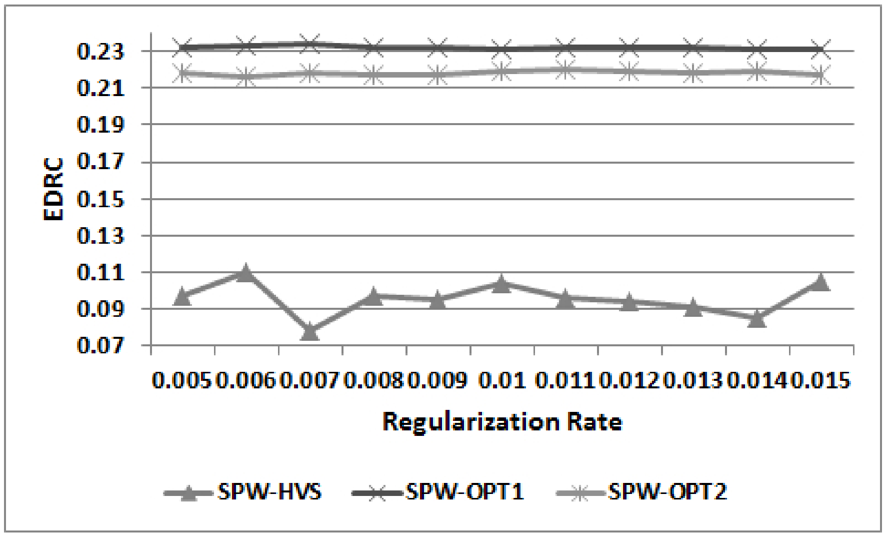


Figure 6.38: EDRC Comparison of Changing Regularization Rate for Sessions

overfitting of the model, while large regularization weights may also hinder quality by underfitting the model.

Chapter 7

CONCLUSION

7.1 Conclusion

Recommender systems are a useful type of information filtering system. This is because they allow for results to be tailored to specific users. Existing work on recommender systems allows for input data in the form of cardinal and ordinal user feedback. However, there is possibility to use a more general form of user feedback, pairwise preferences. Pairwise preferences can be generated from cardinal and ordinal data, but can also be found using implicit user feedback. Additionally, there is the possibility to leverage context data in the form of the offer set of items in a user session.

In this thesis, I offer solutions to operate on pairwise preference data found globally and within session. Additionally, I offer a new evaluation technique that allows for evaluation this type of data. My proposed solutions are in this thesis are outlined below:

Pairwise Preference Model

Cardinal and ordinal user feedback is restrictive because of the finite levels of preferences that can be expressed. Pairwise preferences solves this problem by allowing for any number of levels of user preference. I offer a new model to handle pairwise preference data that borrows the idea of opportunity cost from economics. This model can prediction a preference between two items which can be used to rank items for a user when they need a recommendation. This model can be altered to additionally handle preference found within user sessions.

Leveraging Context Data

Existing work leveraging the context of the offer set of item in a user session has been limited to CCF [75]. However, in this work came with certain limitations such as only allowing for one chosen items with the other items being less preferred which is only binary relevance feedback. This offered the possibility to work on a more general problem for any number of ordinal relevance classes. Additionally, CCF only generated context-independent relevance values. I offered an extension of the opportunity cost that handled any number of ordinal relevance classes and also could generate context-dependent relevance values.

New Evaluation Techniques

Finally, I noticed that there was not an existing weighted measure or metric to handle pairwise preference data. The only existing metric to handle pairwise preferences that induce a partial order was pairwise loss. However, this was not weighted which is desirable to weight more relevant item more heavily than less relevant items. I proposed expected discounted rank correlation or EDRC as a weighted measure that handles partially ordered lists derived from implicit user feedback which is commonly pairwise preferences.

7.2 Future Work

In this section, I discuss areas for possible future work.

Improvements for Efficiency

Looking at the running time for the experiments done in the global problem setting, it is easily seen that there is room for improvement. A possibility for improvement would be to create different models for prediction and learning as seen in [75]. This drastically improves the running and learning times of their approach and could be employed to make my model more efficient.

User Budget

My model assumes that a user has an unlimited budget to take opportunities and does not assume any user constraints (e.g. time or money). It is possible to exploit this information in my model and learning techniques. It would also help to understand why some sessions do not have any user interaction as the user may not have time to watch a 3 hour movies or buy a $50 book.

System Strategy

Another area for future work is having adaptive strategies for the system based on system constraints such as inventory or sponsored items. Research in this area is likely to come with the offering of the RecLab Prize from Overstock.com [54] which encourages research to maximize revenue for the system.

Handling Negative Feedback

My methods for context-aware recommendation do not allow for sessions where no user interaction occurs. This can be seen as negative user feedback, meaning the user is saying that is not does value any of the items being displayed. New methods may be proposed to handle this case and update the model accordingly to lower the value of each item in the session based on the context of the other items.

Further Experimental Evaluation

It is possible to run more experiments using the methods presented in my work. This would include using other datasets and performing a larger scale user study. Additionally, further evaluation can be performed to use other rank correlation measures such as Spearman's ρ or Kendall's τ.

REFERENCES

[1] A. Abdelwahab, H. Sekiya, I. Matsuba, Y. Horiuchi, and S. Kuroiwa. Collaborative filtering based on an iterative prediction method to alleviate the sparsity problem. In *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '09, pages 375–379, 2009.

[2] B. Ackerman and Y. Chen. Evaluating rank accuracy based on incomplete pairwise preferences. In *Proceedings of the 2nd International Workshop on User-centric Evaluation of Recommender Systems and their Interfaces*, 2011.

[3] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, Jan. 2005.

[4] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734 – 749, june 2005.

[5] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*, 2008.

[6] E. Agichtein, S. Lawrence, and L. Gravano. Learning to find answers to questions on the web. *ACM Transactions on Internet Technology*, 4(2):129–162, May 2004.

[7] Amazon. Amazon mechanical turk, May 2012.

[8] A. Ansari, S. Essegaier, and R. Kohli. Internet recommendation systems. *Journal of Marketing Research*, 37(3):pp. 363–375.

[9] J. Bobadilla and F. Serradilla. The effect of sparsity on collaborative filtering metrics. In *Proceedings of the Twentieth Australasian Conference on Australasian Database - Volume 92*, ADC '09, pages 9–18, 2009.

[10] A. Borchers, J. Herlocker, J. Konstan, and J. Reidl. Ganging up on information overload. *Computer*, 31(4):106 –108, apr 1998.

[11] C. Boutilier, R. Brafman, C. Geib, and D. Poole. A constraint-based approach to preference elicitation and decision making. aaai spring symposium on qualitative decision theory, 1997.

[12] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998.

[13] D. Bridge, M. H. Göker, L. McGinty, and B. Smyth. Case-based recommender systems. *Knowl. Eng. Rev.*, 20(3):315–320, Sept. 2005.

[14] K. S. Candan, H. Cao, Y. Qi, and M. L. Sapino. System support for exploration and expert feedback in resolving conflicts during integration of metadata. *The VLDB Journal*, 17(6):1407–1444, Nov. 2008.

[15] L. Chen and P. Pu. Survey of preference elicitation methods. Technical report, SWISS FEDERAL INSTITUTE OF TECHNOLOGY IN LAUSANNE (EPFL, 2004.

[16] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1), Jan. 2004.

[17] P. W. Foltz and S. T. Dumais. Personalized information delivery: an analysis of information filtering methods. *Commun. ACM*, 35(12):51–60, Dec. 1992.

[18] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 1999.

[19] S. Funk. Netflix update: Try this at home, December 2006.

[20] M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies. *Artif. Intell.*, 174(3-4):270–294, Mar. 2010.

[21] GroupLens. Movielens data sets, August 2011.

[22] D. R. Henderson. Opportunity cost, January 2008.

[23] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA, 2000. MIT Press.

[24] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999.

[25] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 194–201, 1995.

[26] Z. Huang, H. Chen, and D. Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):116–142, Jan. 2004.

[27] J. Iijima and S. Ho. Common structure and properties of filtering systems. *Electron. Commer. Rec. Appl.*, 6(2):139–145, July 2007.

[28] IMDB. Johnny depp, June 2012.

[29] K. Jarvelin and J. Kekalainen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, October 2002.

[30] M. Kahng, S. Lee, and S.-g. Lee. Ranking in context-aware recommender systems. In *Proceedings of the 20th international conference companion on World wide web*, 2011.

[31] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[32] K. C. Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical Programming*, 90:1–25, 2001.

[33] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008.

[34] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, August 2009.

[35] Y. Koren and J. Sill. Ordrec: an ordinal model for predicting personalized item rating distributions. In *Proceedings of the fifth ACM conference on Recommender systems*, 2011.

[36] J. Lees-Miller, F. Anderson, B. Hoehn, and R. Greiner. Does wikipedia information help netflix predictions? In *Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications*, ICMLA '08, pages 337–343, 2008.

[37] G. Lekakos and G. M. Giaglis. Improving the prediction accuracy of recommendation algorithms: Approaches anchored on human factors. *Interact. Comput.*, 18(3):410–431, May 2006.

[38] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan. 2003.

[39] N. N. Liu and Q. Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 83–90, 2008.

[40] T.-Y. Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, Mar. 2009.

[41] P. Lops, M. de Gemmis, G. Semeraro, F. Narducci, and C. Musto. Leveraging the linkedin social network data for extracting content-based user profiles. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 293–296, 2011.

[42] F. Lorenzi and F. Ricci. Case-based recommender systems: a unifying view. In *Proceedings of the 2003 international conference on Intelligent Techniques for Web Personalization*, ITWP'03, pages 89–113, 2005.

[43] H. Ma, T. C. Zhou, M. R. Lyu, and I. King. Improving recommender systems by incorporating social contextual information. *ACM Trans. Inf. Syst.*, 29(2):9:1–9:23, Apr. 2011.

[44] T. Mahmood and F. Ricci. Adapting the interaction state model in conversational recommender systems. In *Proceedings of the 10th international conference on Electronic commerce*, ICEC '08, 2008.

[45] T. Mahmood and F. Ricci. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, HT '09, pages 73–82, 2009.

[46] S. Maneeroj and A. Takasu. Hybrid recommender system using latent features. In *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops*, WAINA '09, pages 661–666, 2009.

[47] L. Martinez, R. M. Rodriguez, and M. Espinilla. Reja: A georeferenced hybrid recommender system for restaurants. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, WI-IAT '09, pages 187–190, 2009.

[48] MathWorks. Matlab: The language of technical computing, June 2012.

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

[50] E. Minkov, B. Charrow, J. Ledlie, S. Teller, and T. Jaakkola. Collaborative future event recommendation. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010.

[51] Netflix. Netflix prize, June 2012.

[52] K. Oku, S. Nakajima, J. Miyazaki, S. Uemura, and H. Kato. A ranking method based on users' contexts for information recommendation. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, 2008.

[53] K. Oku, S. Nakajima, J. Miyazaki, S. Uemura, and H. Kato. A recommendation method considering users' time series contexts. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, 2009.

[54] Overstock. Reclab prize on overstock.com, June 2012.

[55] C. Palmisano, A. Tuzhilin, and M. Gorgoglione. Using context to improve predictive modeling of customers in personalization applications. *Knowledge and Data Engineering, IEEE Transactions on*, 20(11):1535 –1549, nov. 2008.

[56] U. Panniello and M. Gorgoglione. A contextual modeling aproach to context-aware recommender systems. In *Proceedings of the 3rd International Workshop on Context-Aware Recommender Systems*, 2011.

[57] U. Paquet, B. Thomson, and O. Winther. A hierarchical model for ordinal matrix factorization. *Statistics and Computing*, 22(4), July 2012.

[58] K. Park, J. Choi, and D. Lee. Iptv-vod program recommendation system using single-scaled hybrid filtering. In *Proceedings of the 10th WSEAS international conference on Signal processing, computational geometry and artificial vision*, ISCGAV'10, pages 128–133, 2010.

[59] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proc. KDD Cup and Workshop*, 2007.

[60] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, 2009.

[61] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, 1994.

[62] F. Ricci, B. Arslan, N. Mirzadeh, and A. Venturini. Itr: A case-based travel advisory system. 2002.

[63] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., 2010.

[64] F. Rossi, K. B. Venable, and T. Walsh. Preferences in constraint satisfaction and optimization.

[65] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Preedings of the 10th international conference on World Wide Web*, 2001.

[66] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 253–260, 2002.

[67] R. Schirru. Topic-based recommendations in enterprise social media sharing platforms. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 369–372, 2010.

[68] E. Schonfeld. With 80 million users, pandora files to go public, February 2011.

[69] U. Shardanand and P. Maes. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1995.

[70] R. R. Sinha and K. Swearingen. Comparing Recommendations Made by Online Systems and Friends. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.

[71] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[72] A. Szwabe, T. Janasiewicz, and M. Ciesielczyk. Hybrid recommendation based on low-dimensional augmentation of combined feature profiles. In *Proceedings of the Third international conference on Computational collective intelligence: technologies and applications - Volume Part II*, ICCCI'11, pages 20–29, 2011.

[73] J. Wang, A. P. de Vries, and M. J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[74] Wolfram. Wolfram mathematica, June 2012.

[75] S.-H. Yang, B. Long, A. J. Smola, H. Zha, and Z. Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, 2011.

[76] Y. Y. Yao. Measuring retrieval effectiveness based on user preference of documents. *Journal of the American Society for Information Science*, 46(2):133, 1995.

[77] E. Yilmaz, J. A. Aslam, and S. Roberston. A new rank correlation coefficient for information retrieval. In *Proceedings of the 31st Annual International ACM SIGIR Conference*, 2008.

APPENDIX A

User Study

Directions to the User

Instructions (please read): In this user study, you will be asked about your preferences for movies given five different queries (e.g. "Leonardo DiCaprio").

- Click and drag movies anywhere between least and most preferred to express differing levels of your preferences
- DO NOT just drag some movies over to most preferred, leaving the rest on least preferred
- If you are unsure of a movie, use genres and actors to make your best judgement

NOTE: Results from irresponible users will be rejected. Please take your time and give your honest preferences. Users only dragging movies to one side or the other will be rejected!

Queries

Table 7.1: Query 1 – “*Tom Hanks actor*”

Forrest Gump
Comedy | Drama | Romance
Tom Hanks, Robin Wright and Gary Sinise
Cast Away
Adventure | Drama
Tom Hanks, Helen Hunt and Paul Sanchez
Saving Private Ryan
Action | Drama | History
Tom Hanks, Matt Damon and Tom Sizemore
Green Mile
Crime | Drama | Fantasy
Tom Hanks, Michael Clarke Duncan and David Morse
Toy Story
Animation | Adventure | Comedy
Tom Hanks, Tim Allen and Don Rickles
Catch Me If You Can
Biography | Comedy | Crime
Leonardo DiCaprio, Tom Hanks and Christopher Walken

Table 7.2: Query 2 – “*Steven Spielberg director*”

Schindler’s List
Biography | Drama | History
Liam Neeson, Ralph Fiennes and Ben Kingsley
Saving Private Ryan
Action | Drama | History
Tom Hanks, Matt Damon and Tom Sizemore
Raiders of the Lost Ark
Action | Adventure
Harrison Ford, Karen Allen and Paul Freeman
Jaws
Thriller
Roy Scheider, Robert Shaw and Richard Dreyfuss
Jurassic Park
Adventure | Sci-Fi
Sam Neill, Laura Dern and Jeff Goldblum
Catch Me If You Can
Biography | Comedy | Crime
Leonardo DiCaprio, Tom Hanks and Christopher Walken

Table 7.3: Query 3 – “*Tom Cruise actor*”

Minority Report
Action | Mystery | Sci-Fi
Tom Cruise, Colin Farrell and Samantha Morton
War of the Worlds
Action | Adventure | Drama
Tom Cruise, Dakota Fanning and Tim Robbins
Rain Man
Drama
Dustin Hoffman, Tom Cruise and Valeria Golino
Mission: Impossible
Action | Adventure | Thriller
Tom Cruise, Jon Voight and Emmanuelle BÃľart
Top Gun
Action | Drama | Romance
Tom Cruise, Kelly McGillis and Val Kilmer
Jerry Maguire
Comedy | Drama | Romance
Tom Cruise, Cuba Gooding Jr. and RenÃľe Zellweger

Table 7.4: Query 4 – “*Leonardo DiCaprio actor*”

Inception
Action | Adventure | Sci-Fi
Leonardo DiCaprio, Joseph Gordon-Levitt and Ellen Page
The Departed
Crime | Drama | Mystery
Leonardo DiCaprio, Matt Damon and Jack Nicholson
Titanic
Adventure | Drama | History
Leonardo DiCaprio, Kate Winslet and Billy Zane
Shutter Island
Drama | Mystery | Thriller
Leonardo DiCaprio, Emily Mortimer and Mark Ruffalo
Catch Me If You Can
Biography | Comedy | Crime
Leonardo DiCaprio, Tom Hanks and Christopher Walken
Blood Diamond
Adventure | Drama | Thriller
Leonardo DiCaprio, Djimon Hounsou and Jennifer Connelly

Table 7.5: Query 5 – “*Robert Zemeckis director*”

Forrest Gump
Comedy | Drama | Romance
Tom Hanks, Robin Wright and Gary Sinise

Cast Away
Adventure | Drama
Tom Hanks, Helen Hunt and Paul Sanchez

Back to the Future
Adventure | Comedy | Sci-Fi
Michael J. Fox, Christopher Lloyd and Lea Thompson

Beowolf
Animation | Action | Adventure
Ray Winstone, Crispin Glover and Angelina Jolie

Contact
Drama | Mystery | Sci-Fi
Jodie Foster, Matthew McConaughey and Tom Skerritt

Who Framed Roger Rabbit
Animation | Comedy | Crime
Bob Hoskins, Christopher Lloyd and Joanna Cassidy