USING META-DATA FROM FREE-TEXT USER-GENERATED CONTENT TO IMPROVE PERSONALIZED RECOMMENDATION BY REDUCING SPARSITY

XU XIAOYING

(B.Eng. (Hons.), South China University of Technology)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF INFORMATION SYSTEMS

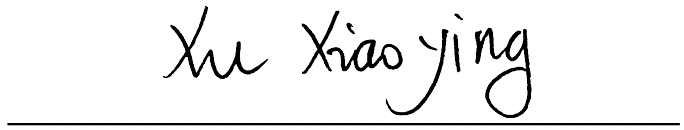
NATIONAL UNIVERSITY OF SINGAPORE

2015

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.



Xu Xiaoying

28 May 2015

ACKNOWLEDGEMENTS

My deepest gratitude goes first and foremost to Professor Anindya Datta, my supervisor, for his constant guidance and encouragement. This thesis could not have been accomplished without his illuminating instruction. The days working with him have become a memorable journey in my life.

Second, I would like to express my heartfelt gratitude to my dear friends and my fellow classmates in NUS who gave me the kindest help. I will never forget the days spent with them.

Last I want to thank my beloved parents and girlfriend, for their loving consideration and great patience in the past few years. I dedicate this thesis to them.

TABLE OF CONTENTS

SUMMARY

Recommender Systems (RS) have become increasingly essential in many domains for alleviating the “information overload” problem, but existing recommendation techniques suffer from the sparsity problem due to insufficient input data.

In this thesis, we aim at extracting and incorporating meta-data from free-text User-Generated Content (UGC) to lessen the effects of sparsity and therefore improve the quality of recommendation. We achieve this goal by conducting three different studies, each of which proposes a recommendation solution that incorporates UGC from different perspectives, and addresses specific problems introduced by data sparsity in different contexts.

In particular, in study one (Chapter 3), we show that adjective features embedded in user reviews are useful for characterizing movie features as well as user tastes. We extend the standard TF-IDF term weighting scheme by introducing Cluster Frequency (CLF) to automatically extract high quality adjective features from user reviews, and incorporate the extracted adjective features into a specific recommendation technique, i.e. Singular Value Decomposition (SVD) to show effectiveness.

In study two (Chapter 4), we show that critic reviews of the items can be used to boost new item recommendation. We collect critic review articles for

corresponding items in recommender system, and employ topic model to quantify the textual content. We adapt Non-negative Matrix Factorization (NMF) to incorporate the topics inferred from the critic reviews for recommendation, aiming at addressing the new item recommendation problem.

Study three (Chapter 5) focuses on extracting functional aspects from user reviews for mobile app recommendation. With the extracted functional aspects, we are able to analyze user requirements at the functional level. We propose a graph-based ranking algorithm to predict new functionalities for users, and devise a competition mechanism to filter redundant recommendations. Our proposed solution is effective in improving stability against data sparsity and increasing the accuracy and diversity of mobile app recommendation.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1. Overview of Recommender Systems

Recommender systems (RS) are well-known artifacts in consumer marketing, having been utilized to great commercial success in iconic technological companies like Amazon, TiVo and Netflix. Commensurate with their market impact, RS technology has enjoyed (and continues to enjoy) much attention from scientists and researchers. Over the past decade, numerous papers have been published, systems have been released and entire top-rated conferences have been established, backed by leading scientific and technological associations, on RS research. Suffice to say, in the domain of data mining, knowledge discovery and information retrieval, recommender systems stand out as one of the most prominent examples of the real-life impact of academic research.

Given the relatively long history of this field, many different paths have been followed to create a variety of RS, albeit with the same end-goal – recommending objects of interest to a user. At a high level, and based on the types of technique and data it uses to generate recommendations, RS may be classified into three main streams, i.e. Collaborative Filtering (CF), Content-Based (CB) and Social-Network-Based (SNB).

Among these recommendation techniques, CF is the most common form

of recommender system used in practice. The basic idea of CF is to recommend items, that similar users (i.e. “neighbors” of the target user) like, to the target user (Resnick et al. 1994; Sarwar et al. 2001). The success of CF is due to its compelling simplicity and high quality of recommendations.

Differing from CF, another stream of recommendation algorithms, i.e. CB recommendation, aims to recommend items similar to what the target user has liked in the past, based on similarities in content(Lops et al. 2011; Pazzani and Billsus 2007).

In recent developments, boosted by the popularity of social network, SNB recommendation has been proposed (Cai et al. 2011; Groh et al. 2012). The intention of SNB is to replace rating-similarity-based neighborhoods in CF with sub-graphs of the user’s social network, motivated by the fact that “people prefer recommendations from people they know” (Bonhard and Sasse 2006).

1.2. Problem Description

In spite of its popularity, RS still faces many challenges (Adomavicius and Tuzhilin 2005). Arguably, the most major and challenging weakness that permeates virtually every flavor of RS is the problem of sparsity. Sparsity refers to the insufficiency of input data into recommendation algorithms.

The most common RS, Collaborative Filtering (CF) systems (powering virtually all commercial systems today), rely heavily on user ratings.

Unfortunately, in most domains studied (movies, books, restaurants etc.), a majority of items turn out to be unrated, resulting in sparse rating matrices (matrices with insufficient data), which adversely impact the quality of recommendations (Adomavicius and Tuzhilin 2005; Su and Khoshgoftaar 2009). The sparsity problem assumes special significance in emerging, high-impact product segments like mobile applications for smart devices, where the cardinality of the underlying domain is of a higher order of magnitude than movies, for example.

The other broad class of RS, Content-Based (CB) systems, suffers not only from rating sparsity, but also from feature, or attribute, sparsity. In the example of movies, the idea is to translate a user's rating into a set of feature preferences. For instance, if User *A* has rated the movie *Argo* highly, the system might assume that the user is expressing a preference for the director, genre and performers of *Argo*. Clearly, this suffers from the same issue of rating sparsity as in CF techniques, but in addition, it must restrict its judgment based on a small number of features typically selected for media objects (e.g., director, genre and performer for movies; singer, songwriter and composer for music; etc.).

The sparsity problem is compounded for Social-Network-Based (SNB) systems, where the quality of SNB approaches is strongly affected by network density. A well-connected user network is vital and essential for such

approaches to ensure the quality of recommendations. However, in reality, user connections are usually very sparse, especially when the social network has been newly introduced into the system.

As a fundamental problem, data sparsity not only lowers the accuracy of recommendations, but also brings up many other subsequent problems in RS such as low coverage, low transparency and low diversity (Adomavicius and Tuzhilin 2005). To address these problems, it is worthwhile to target this issue of data sparsity.

To conclude, although existing recommendation algorithms have achieved some degree of success, there exist substantial opportunities for further improvements. One promising approach to improve existing methods is to alleviate data sparsity by exploring other valuable data. Therefore, the general research question of our studies is: What kind of external data can be used, and how to incorporate such supplementary data into RS, to ameliorate data sparsity?

1.3. Motivation and Research Focus

In this context, our goal is to create general recommendation solutions that would ameliorate problems introduced by data sparsity to improve the quality, including the accuracy, coverage, diversity and transparency of traditional recommendation algorithms.

We embark on this journey motivated by a simple intuition – intelligently

incorporating informative content might allow for gauging the taste of a user, which in turn might allow us to make intelligent data-based estimations of the user's preference for products, thereby reducing sparsity. We start off by exploring possible data to incorporate, and immediately notice an interesting phenomenon: a readily available source of information for many consumer products (movies, books, hotels, electronic products, mobile apps) is User-Generated Content (UGC).

UGC may appear in different forms on the Internet. Currently, there is no standardized definition of UGC. In our context, similar to (Clever et al. 2009), UGC refers mainly to textural content created and published by online users on the consuming end. More specifically, it can be in the form of customer reviews and feedback text, or critic reviews for consumer products.

To achieve our goal, we conduct three different studies, each of which proposes a recommendation solution that incorporates UGC from different perspectives, and addresses specific problems introduced by data sparsity in different contexts. Specifically, the focus of each of the three studies is briefly described below.

In study one, we intend to show that adjective features embedded in user reviews are useful for characterizing movie features as well as users' taste, and can be employed by recommendation techniques to address sparsity and

transparency issues. We employ Part-of-Speech (POS) tagging and introduce Cluster Frequency (CLF) into the traditional TF-IDF term weighting scheme to extract adjective features from external user reviews, relieving the problem of diverse vocabulary, and balancing the representativeness and generalizability of the extracted features. We also incorporate the extracted adjective features into a specific recommendation technique, i.e. Singular Value Decomposition (SVD), to illustrate the effectiveness of using adjective features.

In study two, we propose a novel content-based recommendation solution. A distinct feature of our method is that it incorporates the topics inferred from external critic reviews to boost recommendations for new items. We employ an advanced semi-supervised topic modeling approach, i.e. Partially Labeled Dirichlet Allocation (PLDA), which is able to uncover globally-shared latent topics, as well as topics under each well-structured item attribute, to learn and infer the topic distribution of critic reviews. We also adapt Non-negative Matrix Factorization (NMF) to our context by redefining the error function to fully utilize user ratings and the topic distribution of critic reviews. The topics inferred from critic reviews are better representations of the items, since they cover more characteristics of the items and reflect more aspects of user tastes. By fully utilizing user ratings and the inferred topics, our method alleviates the dependency on user ratings and enables high-quality recommendations, even in cold-start settings with new items. The adoption of NMF lowers the dimension

of the original rating matrix, which contributes to higher efficiency.

In study three, we propose to analyze users' requirements at the functional level, with the objectives of avoiding recommending redundant apps, and helping users find better apps that are not just similar in nature. A main feature of our approach is mining textual user reviews. We develop a crawler to collect user reviews of each app from App Stores, and propose aspect identification techniques to mine functionality-related aspects from these reviews. Moreover, we propose a two-stage graph-based ranking algorithm to predict new functionalities for users, and come up with a competition mechanism to intelligently filter out redundant apps. By using app functionality as the unit of analysis, we successfully improve system stability against data sparsity, and increase recommendation accuracy and diversity.

1.4. Contribution

Our research seeks to contribute to both academics and practitioners in the field of RS by addressing data sparsity. Specifically, by incorporating User-Generated Content (UGC) from different perspectives, our studies address specific problems (i.e. accuracy, diversity, coverage, transparency) in different contexts. To summarize, the main contributions that make our studies important are as follows.

Firstly, our studies prove that different kinds of meta-data from UGC can

be extracted and incorporated to facilitate recommendations. Although UGC is a promising and valuable source of information, the use of textual UGC in designing RS has received scant attention from scientists. There exist a few papers regarding the incorporation of free-text user reviews to perform recommendations. But while they focused on the sentiment of UGC, they ignored the meta-data embedded within. Our studies are among the first to consider extracting meta-data from UGC for the purposes of recommendation.

Second, we propose to adapt feature extraction techniques to our context to extract high quality meta-data for the purposes of recommendation. For example, we introduce Cluster Frequency (CLF) into the traditional TF-IDF term weighting scheme, extracting not-too-general and not-too-special adjective features. We also adapt Partially Labeled Dirichlet Allocation (PLDA) to model critic reviews and represent movies at a higher and more abstract level. In addition, we propose an effective approach aimed at extracting functional aspects of mobile apps from user reviews. Our adaptions of feature extraction techniques have implications for both UGC and RS research.

Third, we propose several approaches to incorporate UGC into RS by utilizing the extracted meta-data and user ratings. For example, we adapt Singular Value Decomposition (SVD) to represent user tastes and movie characteristics as feature vectors. We also adapt Non-negative Matrix Factorization (NMF) to model user topic preferences and movie topic

distributions. We come up with an effective approach, i.e. a two-stage graph-based ranking method and a completion mechanism, to maximize the utility of functional aspects extracted from user reviews in mobile app recommendations. Our studies have implications for RS research attempting to incorporate textual content. We also aim to fill the gap between UGC and RS research.

1.5. Organization of Thesis

The opening chapter provides the context and motivation of our research, as well as a brief introduction to the three studies included in this thesis. Chapter 2 reviews the literature on three main streams of RS and current trends of using UCG in RS. Chapter 3 describes the first study that uses adjective features from user reviews to address sparsity and transparency issues in RS. Chapter 4 describes the second study that uses critic reviews to boost new item recommendations. Chapter 5 describes the third study that identifies functional aspects from user reviews for functionality-based mobile app recommendations. Finally, Chapter 6 summarizes the work in this thesis and outlines future directions.

CHAPTER 2. LITERATURE REVIEW

Substantial research has been conducted on recommendation algorithms, mostly belonging to three main streams, i.e. Collaborative Filtering (CF) approaches, Content-Based (CB) approaches and Social-Network-Based (SNB) approaches. Our studies belong to the family of CB approaches, as they incorporate extracted meta-data from User-Generated Content (UGC) into RS. We also borrow some advanced techniques from CF models to utilize user ratings as well as the extracted meta-data. In this chapter, we review the work on three main streams of RS research in general. A survey on recent RS research using UGC is also included.

2.1. Collaborative Filtering (CF) Recommendation

CF has been explored in-depth in the past ten years, and represents the most popular recommendation algorithm, owing to its compelling simplicity and excellent quality of recommendations. Typically, CF techniques can be classified into three categories: memory-based CF, model-based CF and graph-based CF.

2.1.1 Memory-Based CF

The most common approaches to CF are memory-based, which means that the entire user-item rating matrix is used to generate predictions. User-based CF (Resnick et al. 1994) is one of the earliest methods of memory-based CF, where

the basic idea is to recommend items that similar users (i.e. “neighbors” of the target user) like, to the target user. This approach is simple and easy to implement, but it has difficulty in generating recommendations for new users. Another type of memory-based CF, Item-based CF (Sarwar et al. 2001), was later proposed. In contrast to User-based CF, Item-based CF recommends items highly correlated with those items liked by the target user. Item-based CF is able to address the problems associated with new users, and achieves higher scalability and accuracy.

Memory-based CF can be implemented easily and new data can be added incrementally at little cost. However, memory-based CF has high space complexity, and it is unable to handle large datasets. These inadequacies can be addressed by model-based CF.

2.1.2 Model-Based CF

Compared to memory-based CF, model-based CF does not require the entire rating matrix, but learns to recognize complex patterns to train models based on training data (which is a small subset of the whole dataset), and then uses the trained models to make predictions for CF tasks with real-world data.

Latent factor models, such as Probabilistic Matrix Factorization (PMF), comprise an alternative approach to CF, by transforming both items and users to the same latent factor space, which explains ratings by characterizing both

users and items on the factors that are automatically inferred from user feedback (Koren and Bell 2011). Examples include Neural Networks (Salakhutdinov et al. 2007), Probabilistic Latent Semantic Analysis (Hofmann 2004), Latent Dirichlet Allocation (Blei et al. 2003) and Singular Value Decomposition (SVD) (Paterek 2007).

Memory-based CF and model-based CF tend to recommend well-known items and give less weight to the new items, which lowers the diversity of recommendations. This problem can be addressed by graph-based CF.

2.1.3 Graph-Based CF

Graph-based CF represents data as a graph, where users and items are represented as nodes and edges, capturing the interaction between users and items. Aggarwal et al. (1999) proposed a graph-theoretic CF approach in which the similarity between two users is computed based on their shortest distance in the graph. When predicting the rating of a user for a new item, the shortest directed paths from this user to other users who have also rated this item are obtained, and their ratings are used. Huang et al. (2004) used the number of paths between the user and the item to estimate the user's preference on this item. Pucci et al. (2007) adapted Google's PageRank algorithm for ranking searching results, and proposed the ItemRank approach that ranks a user's preference towards items, by computing the probability that this user will visit

the item nodes in a random walk of the graph, where the edges between item nodes connect the items commonly rated by users. Proposed by Google and having been applied in the YouTube video suggestion engine, Baluja et al. (2008) also employed a random walk model on the video co-view graph to generate personalized video suggestions for users.

Graph-based CF has the advantage of discovering new items, improving the novelty of recommendations, but it faces the problem of extremely high computational expenses.

Despite its popularity, CF recommendation has many problems. The quality of CF largely relies on user ratings that are usually very sparse in reality. Moreover, CF usually works as a black box without offering much transparency, which may lower user trust. Lacking the ability to recommend new items is another well-known inadequacy of CF. CB recommendation is a different approach that is able to address the transparency issue and the problem of new items.

2.2. Content-Based (CB) Recommendation

CB recommendation aims at recommending items similar to what the target user has previously liked. A typical CB RS constructs a profile, which is a structured representation of interests for every user, by analyzing the description of items previously rated by this user. The recommendation process matches up the user

profile against the attributes of new items (Pazzani and Billsus 2007). The nature of the CB approach enables new item recommendations, since it does not require any user preference data of the new items. CB recommendation can also capture taste aspects of users and explain how the recommender system works, by explicitly listing content features or descriptions that cause an item to occur in the list of recommendations, while CF is unable to explore detailed aspects of users' taste, since the data only comes from users' ratings.

CB RS, in the domain of consumer products (books, movies, mobile apps, etc.), usually uses well-structured attributes to represent items. For example, the genre, directors and actors of movies are commonly used in movie RS (Gantner et al. 2010; Maneeroj and Takasu 2009; Manzato 2012). Such systems have a natural limit on the number and type of features that are associated with the items recommended. Research has found that features assigned to items are insufficient to define distinguishing aspects of items that turn out to be necessary for the elicitation of user interests (Lops et al. 2011).

In CB systems, the user profile learner is a core component. Many existing methods regard user preference as a binary attribute (i.e. like or dislike) and therefore, the recommendation problem can be treated as a problem of classification. A series of classification learners have been applied to learn user profiles, including Decision Tree(Bouza et al. 2008), Bayesian classifier (Gutta et al. 2000), SVM (Xu and Araki 2006), Neural Network (Christakou et al. 2007)

etc. These methods have been criticized due to their high complexity and poor interpretability. They also fail to utilize user ratings.

To summarize, CB approaches show promise in addressing new item and transparency problems, but they are limited by inadequate item features and inefficiency in utilizing user ratings. Possible extensions to CB systems can seek other informative data to incorporate and propose effective methods to utilize such data, as well as user ratings, which our studies address.

2.3. Social-Network-Based (SNB) Recommendation

With the explosion of social network sites, e.g. Facebook and Twitter, another type of recommender systems, i.e. social recommender systems, has gained popularity. The basic idea of social recommender systems is to replace rating-similarity-based neighborhoods in CF with sub-graphs of user's social networks, motivated by the fact that "people prefer recommendations from people they know" (Bonhard and Sasse 2006).

(Said et al. 2010) investigated a movie recommender system providing underlying social networks, and proved that the quality of recommendations could be improved by utilizing user-user relations. A trust-based network embedded in a social network offers an alternative approach to overcome the data sparsity problem in CF. Golbeck (2006) used a Probabilistic Matrix Factorization (PMF) framework that incorporates the user-rating matrix as well

as users' social trust network to generate recommendations, which outperforms the CF approach, especially when the ratings are sparse. Jamali and Ester (2010) incorporated the trust propagation mechanism into the matrix factorization technique, leading to substantial increase in recommendation accuracy. Graph-theoretic technology has also been applied to analyze social networks for recommendation. Wang et al. (2010) proposed to use a graph random walk model to capture users' similarity in social influence, and applied Singular Value Decomposition (SVD) to predict users' opinions.

Social recommender systems are a new trend that deserves further exploration. However, similar to traditional recommender systems, SNB approaches also suffer from the sparsity problem. The quality of SNB recommendations is strongly affected by the network density, which is very sparse in reality.

2.4. User-Generated-Content (UGC) in Recommendation

To address the challenges of RS, an increasing amount of research has recently started to pay attention to UGC. UGC can be found in abundance on online review platforms and forums. Such content is valuable information that covers more item features and contains consumer opinions.

Lately, there has been much recent interest in a specific kind of UGC, i.e. tags. Tags are generated by users who collaboratively annotate and categorize

resources of interest with freely chosen keywords (de Gemmis et al. 2008). Several methods have been proposed for incorporating tags within CB recommendations. Diederich and Iofciu (2006) represented the user profile in the form of a tag vector; each element indicates the number of times a tag has been assigned to a document by that user. Michlmayr (2007) proposed different strategies to build tag-based user profiles, which were used to produce music recommendations. Wei et al. (2011) proposed a unified framework for recommendations, by modeling the quaternary relationship among users, items, tags and ratings as a 4-order tensor and performed a multi-way latent semantic analysis.

Compared to descriptive attributes typically used in CB RS, tags cover more features of items and are more comprehensible to users. This is also demonstrated in the results reported (Sen et al. 2009). However, since tags are voluntarily and freely provided by users, problems such as the unwillingness to tag and diverse vocabulary can easily arise (Lops et al. 2011). As discussed earlier, the sparsity of ratings is a challenge for rating-based recommendations; here the problem of sparsity is exacerbated in the tag space.

There exists another stream of research that reports on the incorporation of free-text user reviews to perform recommendations, almost all of which employ opinion mining and sentiment analysis techniques to factorize user reviews and then infer user preferences. Aciar et al. (2006) defined an ontology to represent

user reviews, after which an overall rating was aggregated from opinion quality and product quality inferred from user reviews. Jakob et al. (2009) proposed to mine user opinions from free-text movie reviews as supplementary data to user ratings in CF recommendations. Through estimating the reviewer's weight preferences over features, Chen and Wang (2013) constructed an implicit preference network of users, and used this network to generate recommendations. Ganu et al. (2013) employed sentiment analysis to derive a text-based rating from the review body, aimed at improving the quality of restaurant recommendations.

While existing works incorporating UGC in RS have shown promise in alleviating data sparsity problems, there exists substantial opportunities for future research. Our studies follow different routes by extracting and incorporating meta-data of UGC that has received scant attention from RS researchers.

CHAPTER 3. STUDY ON ADDRESSING SPARSITY AND TRANSPARENCY ISSUES IN RECOMMENDER SYSTEMS BY USING ADJECTIVE FEATURES FROM USER REVIEWS

3.1. Introduction

In this study, we aim to create a general approach that would ameliorate the sparsity and transparency issues in RS. It is important to understand that our intent is not to create a completely new recommendation algorithm; rather, our goal is to explore new item features and corresponding techniques for obtaining and incorporating such features, alleviating the effect of rating sparsity and enhancing transparency to significantly improve existing methods.

We are motivated by a simple intuition – representing user interests with plenty of item features might allow us to intelligently translate users' sparse ratings at the item level into detailed feature preferences, thereby reducing the effect of rating sparsity. This may also allow us to explain the rationale of recommendations to users by explicitly listing out relevant item features. In the above example, even if Tom and Jerry have no co-rated movies, after translating their ratings into feature preferences, e.g., "romantic", we can still recommend romantic movies to both of them with the explanation of "liking romantic movies" by relation. It is fair to note that some Content-Based (CB) RS have

the similar idea of using item attributes to represent user interests; however, these methods restrict their judgment based on a small number of structured attributes typically selected for the items (e.g., director, genre and performer for movies; singer, songwriter, composer for music; etc.). Obviously, users' taste aspects extend beyond these limited number of item attributes (Lops et al. 2011), and many more item features are needed in order to comprehensively and accurately capture their taste aspects.

We note that a wealth of information is available from reviews that could possibly be used to enhance the recommendation process. In this study, we focus on one specific kind of information from user reviews, namely, adjective features. While the intent is to incorporate various other types of data from reviews in future work, adjectives represent a particularly attractive feature used in recommendations. When asked to reveal why they like or dislike something, people often use adjectives to explain their preference. For instance, when asked why he/she likes the movie *Titanic*, a user's answer often includes words such as "romantic", "moving", "astounding", "beautiful" or "sad" – all being adjectives. These features truly reflect users' perception and can be found in abundance in user reviews, but this aspect remains unexplored in recommendation research.

Therefore, in this study, we incorporate adjective features extracted from external user reviews in addition to ratings, into the recommendation process to

generate more accurate and more explainable item recommendations, as well as user recommendations. To automatically extract adjective features from user reviews, we employ well-understood part-of-speech (POS) tagging methods. However, we quickly discover that many adjectives are not helpful in discriminating between tastes, i.e., some adjectives are too general to be adequately representative of users' tastes (lack of *representativeness* e.g., "good"), while others are too specific to capture users' general taste aspects (lack of *generalizability* e.g., "unsinkable" in the reviews of *Titanic*).

We tried to search for existing solutions, but we noticed that existing works on adjective extraction and term weighting were restricted and could not be perfectly addressed. Therefore, we propose our own approach, by extending the traditional TF-IDF term weight (Cohen 1995) to TF-IDF-CLF by introducing another unsupervised term weight measure, *Cluster Frequency* (CLF). Unlike other supervised term weighing methods, e.g. (Lan et al. 2009), the newly introduced CLF measure is able to consider implicit item aspects not captured by pre-defined categories, and it also helps balance the representativeness and generalizability of the extracted features.

Although adjective features can be utilized by different recommendation techniques, to make for easier illustration of the effectiveness of our idea, we incorporate the extracted adjective features into one specific recommendation technique, i.e. Singular Value Decomposition (SVD) (Paterek 2007), and then

construct item feature vectors and user feature vectors to generate more accurate rating predictions and explainable recommendations of higher quality by listing adjective features that correlate to the recommended item for the target user. We call this integrated method the Adjective Feature Vector (AFV) method. The result of our work makes substantial advances over extant recommendation techniques. In particular, our method reduces prediction errors from state-of-the-art rating-based methods by 12.42%, in extreme rating-sparse settings. It also outperforms the tag-based method by reducing prediction errors by 11.27% in item recommendations, increasing its interest similarity by 7.14% in user recommendations, and retaining full item and user space coverage. The results also prove our method effective for providing recommendation explanations.

The rest of this chapter is organized as follows: Firstly, we review works related to our study. Then we present the integrated recommendation architecture, including detailed descriptions of each component. The rest of this chapter presents the experiment and results. A summary of the study is given in the conclusion.

3.2. Related Work

Our proposed method extracts adjective features from user reviews for the purpose of both item and user recommendations by adapting keyword extraction techniques and Singular Value Decomposition. In the following, we review

works related to our proposed method.

3.2.1. Adjective Extraction

There are some prior works that extract and incorporate adjectives. For example, Harb et al. (2008) focused on extracting positive and negative adjectives for opinion mining by considering domain knowledge. Voll and Taboada (2007) proposed to determine the positive or negative polarity of text by assigning different weights to adjectives based on their relevance to the object being evaluated. Virtually all these works seek to select adjectives with clear positive or negative polarity (e.g. “good”, “excellent”, “bad”, “poor”) for sentiment analysis, but such general adjectives lack discriminating power and hence are not suitable for representing item features. Middleton et al. (2004) highlighted the importance of selecting representative terms that are “not too common and not too rare”, but did not propose effective solutions above those of removing term suffixes and filtering stop words.

To automatically extract adjectives for our purpose, we need to estimate the weights of each candidate term in the text. There exist some approaches on supervised term weighting, e.g. (Lan et al. 2009), which was designed for text classification, i.e. to determine the likelihood of a term belonging to a pre-defined category. Such methods rely on the limited number of pre-defined categories, but fail to consider terms that help discriminate other implicit item

aspects, and therefore may not be applicable in our context. Our method uses an unsupervised measure, i.e. CLF, which is able to give more weight to those adjectives that help discriminate self-formulated item clusters without being restricted by the limited number of pre-defined categories.

Therefore, our method differentiates itself from existing methods by balancing the representativeness and generalizability of extracted features, and by finding terms that have better discriminating power in many implicit item aspects, rather than a small number of explicit categories.

3.2.2. Singular Value Decomposition (SVD)

SVD is well established for identifying latent semantic factors in the domain of natural language processing (Deerwester et al. 1990). SVD is also a well-known method for matrix factorization, that provides the best lower-ranked approximations of the original matrix. Models that induce SVD to reduce the dimensionality of sparse user-item rating matrices for collaborative filtering have gained popularity due to their accuracy and scalability. SVD models map both users and items to a joint latent factor space having *f* dimensions, and user-item interactions are modeled as inner products in that space. Accordingly, each item *i* is represented as a vector $q_i \in R^f$, in which the elements measure the extent to which an item *i* possesses those factors. Similarly, each user *u* is represented as a vector $p_u \in R^f$ and the value of each element measures the

extent to which the user *u* possesses those factors.

Our proposed method adapts the original SVD to integrate the adjective features extracted from user reviews, with the purpose of addressing sparsity and transparency issues.

3.3. Intuition and Overview

While our approach is general and can be used to recommend any consumer item, we chose a specific domain for the purposes of illustration. Given that the most studied consumer domain, in the context of recommendations, is that of movies, we will henceforth use the movie domain to present our technique. In other words, we will present our method to recommend movies and users with similar interests to the target users.

The general goal of the recommender system is to select the objects that may be of interest to a user. Based on the types of objects it recommends, our method is intended for two tasks: *item recommendation* and *user recommendation*.

For item recommendation, we are interested in predicting ratings for movies new to users, and recommending the movies with the highest predicted ratings to them, together with reasonable and personalized explanations to improve the transparency of the logic in recommendations. Noting that the number of descriptive attributes that are commonly used in content-based movie

recommendation (e.g., actor, director) are limited and insufficient, we automatically extract adjective features from external user reviews (available in abundance in review systems like IMDb[1], and Rotten Tomatoes[2]) to define distinguishing aspects of items and of users' tastes, which are able to truly reflect the users' perception towards movies on a higher and more abstract level. For example, the adjective features extracted from user reviews of *Titanic* can be "romantic", "sad", or "astounding". We predicted the rating of *Titanic* for a user by estimating to what extent *Titanic* is romantic, sad or astounding, and how much the user likes romantic, sad or astounding movies.

For user recommendations, we intend to identify users with common interests so that the connections among users can be expanded. By applying our method, this task can be performed by estimating the similarity between users in terms of each taste aspect characterized by adjective features, and recommending similar users to a given user, together with explanations. For the example mentioned above, for any two users, their similarity is calculated by estimating the extent to which they share the same interest in romantic, sad or astounding movies.

Our method addresses the problem of rating sparsity by decomposing a singular user rating into multiple dimensions explicitly characterized by

---

1 http:// www.imdb.com

2 http:// www.rottentomatoes.com

extracted adjectives, and then translating a small number of user ratings into a larger number of feature preferences, which allows us to better understand users' interests, and to pick out their preferred items more accurately through each of their preferred features, therefore alleviating the problem of item-level rating sparsity. In addition, by explicitly listing adjective features that cause items to be recommended, we are able to explain the logic of recommendation intuitively to users, with the objective of addressing the transparency problem.

3.4. Solution Details

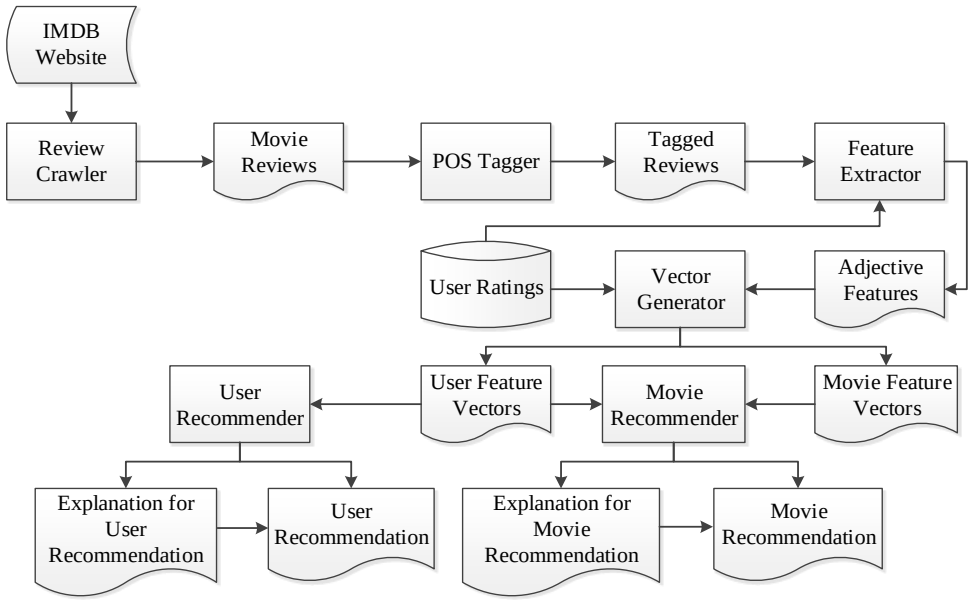


Figure 3.1. Recommendation Architecture

The overview of our movie recommendation architecture is shown in Figure 3.1, where the rectangles represent the components we have designed and implemented to realize our recommendation engine. There are six such components: *review crawler*, *POS tagger*, *feature extractor*, *vector generator*,

item recommender and *user recommender*. Specifically, we introduce Cluster Frequency (CLF) into the feature extractor, which is essential for extracting high-quality adjective features. We incorporate the extracted adjective features into a specific recommendation technique, the Singular Value Decomposition (SVD) (Paterek 2007), and apply stochastic gradient descent optimization to construct movie feature vectors and user feature vectors in the vector generator. We take into account the partial effects of the adjective features causing the item to be recommended in the recommender enabling us to offer explanations for recommendations. Applying this architecture, we incorporate adjective features extracted from IMDb user reviews, as well as user ratings, into the recommendation task, addressing sparsity and transparency issues. More details of each component will be introduced in the following sections of this chapter.

3.4.1. Review Crawler

We obtain user reviews of movies from a reputable external source, i.e. IMDb (the ***Internet Movie Database***). IMDb is one of the most popular online databases for movie information, with over 100 million unique users each month. IMDb also offers a platform for users to review movies, and allows other users to indicate whether they found certain reviews useful. Figure 3.2 shows one user review of *Titanic* on the IMDb website.

Figure 3.2. IMDb User Review Page

To obtain reviews for each movie, we use a web crawler to collect user reviews from the IMDb website. In order to get high-quality reviews, we choose the "Best" filter offered on this website, which ranks the reviews according to the number of users who found the review useful, in descending order. Then we crawl the first 4 pages of user reviews (10 reviews per page) for each movie, and extract the review content from the webpages.

3.4.2. POS Tagger

After obtaining user reviews for each movie, we employ the Stanford POS tagger (Toutanova et al. 2003) to assign parts of speech to each word within the reviews, such as nouns, verbs, adjectives etc. Since we intend to extract adjective features, we keep only adjectives in the reviews. Taking the first paragraph of the review in Figure 2 as an example, after POS tagging, only the

following words remain:

different good great boring clich é beautiful sad

3.4.3. Feature Extractor

This component extracts adjective features from tagged user reviews. Firstly, we assign a weight for each adjective term in the reviews. In the domain of information retrieval, the term weight is a measure of how important a word is in a document. TF-IDF is a very commonly used term weighting scheme. The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times *t* occurs in *d*. Document frequency df_t is defined as the number of documents in the collection that contain a term *t*. The inverse document frequency idf_t of a term *t*, which indicates the term's discrimination power, is defined as:

$$idf_t = \log \frac{N}{df_t} .$$

where *N* is the total number of documents. The TF-IDF term weight for term *t* in document *d* is given by:

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t .$$

We regard the collection of all reviews of a movie as a document. The TF-IDF weight for every word in the reviews can be easily obtained. While features extracted by TF-IDF weight are representative of movie characteristics, they are often tainted by two issues: (a) they may be too specific and might not serve as

generalizable or common characteristics across similar movies, e.g. the word "unsinkable", which has very high TF-IDF scores in the reviews for *Titanic*, is too specific, since we are unlikely to find other movies related to "unsinkable", thus it is unsuitable for representing users' tastes; and (b) they may fail to include some general features that are good for exposing user taste aspects, e.g. when extracting features from the reviews for *Titanic*, the word "sad" may have high TF scores but low IDF scores, therefore resulting in relatively low TF-IDF scores; however, "sad" is a good feature, since it accurately reflects a key user perception towards this movie. In addition to *generality*, as discussed above, the *representativeness* of the extracted features is also important, e.g. the word "good" is too general, such that we cannot use it to represent user preferences. In order to balance representativeness and generalizability, we introduce another term weight measure into TF-IDF, the Cluster Frequency (CLF), to measure how common a word occurs across a cluster of documents similar to a particular document. We get a cluster of similar movies for a given movie and accordingly, all movie reviews in the cluster will be used for calculating the CLF.

For example, if we find a cluster of similar movies for *Titanic*, they may share some common characteristics of tragedy and the word "sad" would have a high frequency across the reviews. By introducing CLF, the term weighing of the word "sad" is higher, and therefore more likely to be extracted. Since we also give importance to the TF-IDF weight, those words that are too general (e.g.

"good") will be filtered out.

Next, we will describe how we identify these clusters of similar movies. The similarity between movies can be computed by either of the following two approaches. First, we can apply item-based CF (Sarwar et al. 2001) and use cosine similarity to compute the distance between two movies based on users' co-rating patterns:

$$\cos(\vec{r_i}, \vec{r_j}) = \frac{\vec{r_i} \times \vec{r_j}}{\left\|\vec{r_i}\right\| \left\|\vec{r_j}\right\|} = \frac{\sum_{u \in U_{i,j}} r_{u,i} r_{u,j}}{\sqrt{\sum_{u \in U_i} r_{u,i}^2 \sum_{v \in U_j} r_{v,j}^2}} \ .$$

where $U_{i,j}$ denotes the set of users rating both movie *i* and movie *j*; U_i denotes the set of users rating movie *i*; and U_j denotes the set of users rating movie *j*. For each movie *i*, we select the top *M* movies having the highest cosine similarity scores as a group of similar movies. We denote this approach as *rating-based clustering*.

Second, noticing that the item-based CF approach heavily depends on user ratings and may not produce good results if ratings are sparse, we also employ the Topic Modeling approach, which is purely based on reviews and eliminates the dependency on user ratings. Latent Dirichlet Allocation (LDA) (Blei et al. 2003) is a generative probabilistic model for collections of discrete data such as text corpora. Since we regard the collection of all reviews of a movie as a document, by applying LDA, each document corresponding to each movie can

be represented as a multinomial distribution over latent topics, where each topic is characterized by a distribution over words. We apply Kullback–Leibler (KL) divergence, which is a non-symmetric measure of the difference between two probability distributions, to calculate the divergence between movie *i*'s topic distribution, P_i, and movie *j*'s topic distribution, P_j:

$$D_{KL}(P_i \parallel P_j) = \sum_{l \in LatentTopics} P_i(l) \ln \frac{P_i(l)}{P_j(l)} .$$

where $P_i(l)$ denotes the probability that movie *i* belongs to the latent topic *l*. For each movie *i*, we select the top *M* movies having the smallest KL divergence, as a cluster of similar movies. We denote this approach as *review-topic-based clustering*.

Unlike supervised term-weighting approaches, in our method, the clusters of similar movies are self-generated without having to rely on the limited number of pre-defined categories, therefore our method is better in discovering terms that have high discriminating power in implicit item aspects not captured by pre-defined categories.

After getting the cluster of similar movies for a movie *i*, the CLF weight of the term *t* in the reviews of *i* can be computed, by counting the number of movies in the cluster whose reviews contain term *t*. Finally, the integrated TF-IDF-CLF term weighting scheme is given by

$$tf\text{-}idf\text{-}clf_{t,i} = tf_{t,i} \times idf_t \times clf_{t,i}^{\lambda_1} \ .$$

where λ_1 is a parameter indicating the weight which is put in the CLF. For each movie, the adjective features are extracted from reviews by selecting the top *K* adjectives having the highest TF-IDF-CLF weights, and then passed to the vector generator.

3.4.4. Vector Generator

Table 3.1. Movie and User Feature Vectors

	romantic	sad	astounding	spectacular	scary
Titanic	0.50	0.40	0.10	-	-
Spider-Man	0.20	-	-	0.30	0.50
User *A*	0.10	0.03	0.50	0.40	0.40
User *B*	-0.10	0.02	0.50	0.30	0.40

After getting the extracted features of each movie, we represent each movie, as well as each user, in the form of feature vectors. Specifically, each movie *i* is represented as a vector Q_i, in which each element is associated with one of its features. The values of the elements measure the extent to which the movie *i* possesses those features. Similarly, each user *u* is represented as a vector P_u, and the elements associated with the features of all movies. The values of the elements measure the extent to which user *u* likes those features. For example, let us assume that we have only two movies in the system, i.e. *Titanic* and *Spider-Man*, and for each movie, we extracted 3 features: from user reviews, the movie feature vectors and the user feature vectors. The results for two given

users are shown in Table 3.1.

Similar to the latent factor model, we included the baseline predictors to estimate the non-interaction effects from users and movies respectively (i.e. $udev_u$ and $idev_i$). A predicted rating of movie *i* for a user *u* is given by:

$$\hat{r}_{u,i} = \mu + udev_u + idev_i + \sum_{f \in F(i)} e_{u,f} e_{i,f} \quad . \tag{3.1}$$

where μ denotes the overall average rating; $udev_u$ and $idev_i$ indicate the observed deviations of user *u* and item *i* respectively from μ; *F(i)* denotes the set of features belonging to the movie *i*; $e_{u,f}$ is the value of feature *f* in user *u*'s feature vector P_u; and $e_{i,f}$ is the value of feature *f* in movie *i*'s feature vector Q_i.

We employ a stochastic gradient descent optimization adapted from Regularized Singular Value Decomposition (RSVD), which was proposed by (Funk 2006) and has been successfully applied by many others (Koren 2008; Paterek 2007), to estimate the values of the elements for both movie feature vectors and user feature vectors, as well as the baseline predictors. For each item *i* we set the initial value:

$$idev_i = \frac{\sum_{u \in R(i)} (r_{u,i} - \mu)}{\lambda_2 + |R(i)|} \quad .$$

And then for each user, *u*, we set the initial value:

$$udev_u = \frac{\sum_{i \in R(u)} (r_{u,i} - \mu - idev_i)}{\lambda_3 + |R(u)|} \quad .$$

where *R(i)* denotes the set of users who rated item *i*; *R(u)* denotes the set of items rated by a user *u*; μ is the overall average rating; and λ_2 and λ_3 are regularization parameters. For each element in movie feature vectors and user feature vectors, we assign an initial value *s*. For each given rating $r_{u,i}$ in the training set, a predicted rating $\hat{r}_{u,i}$ is given by Equation 3.1, and the associated prediction error is defined as:

$$err_{u,i} = r_{u,i} - \hat{r}_{u,i} \ .$$

Then the model parameters are learnt by minimizing the regularized squared error:

$$\min_{udev_u, idev_i, e_{u,f}, e_{i,f}} H(udev_u, idev_i, e_{u,f}, e_{i,f}) =$$
$$\sum_{u \in U, i \in I} \{err_{u,i}^2 + \lambda_4 [udev_u^2 + idev_i^2 + \sum_{f \in F(i)} (e_{u,f}^2 + e_{i,f}^2)]\}.$$

where λ_4 indicates the extent of penalizing the magnitudes of the parameters to avoid over-fitting.

We employ gradient descendent as described in (Funk 2006) to update the baseline predictors and the values of feature vector elements, by moving in the opposite direction of the gradient. We iterate the updating process through the training dataset, until the prediction errors in the validation dataset stop decreasing.

3.4.5. Movie Recommender

With the movie feature vectors and user feature vectors, we can easily predict a rating for a particular user for a given movie, using Equation 3.1. In order to recommend movies to a user, we can predict the ratings of all movies unknown to him, then rank these movies according to the predicted ratings, and recommend the top *N* movies with the highest predicted ratings.

One of the key features of our method is that in addition to providing recommendations, we provide explanations as well. We do this by explicitly listing features that the user likes, and suggesting a movie in the list of recommendations. For each movie *i* in a user *u*'s recommended list, $f \in F(i)$ is one feature in movie *i*'s feature vector, $e_{i,f}$ is the value of feature *f* in movie *i*'s feature vector, and $e_{u,f}$ is the value of feature *f* in user *u*'s feature vector. The product of these two values $e_{u,f} \cdot e_{i,f}$ is the partial interaction effect regarding feature *f*, and measures the extent to which feature *f* contributes to recommend movie *i* to user *u*.

Therefore, we rank all the features that the user likes, i.e. the features with positive values in the user vector and in $F(i)$, according to the partial interaction effect, and provide the top *K* features having the highest products in addition to the recommended movie *i*, as an explanation for the recommendation. Using the aforementioned example, if the movie *Spider-Man* is recommended to user *B*,

we can then obtain the partial interaction effect regarding each feature of *Spider-Man*, as shown in Table 3.2. If we only provide the top feature as the explanation to user *B* for recommending this movie, the feature "scary" is selected, which has a positive value in the user vector, and the highest partial interaction effect.

Table 3.2. User-Item Partial Interaction Effect

	romantic	spectacular	scary
$e_{B,f}$	-0.1	0.3	0.4
$e_{Spider-Man,f}$	0.2	0.3	0.5
$e_{B,f} \times e_{Spider-Man,f}$	-0.02	0.09	0.2

Since the values of these features differ across different users' feature vectors depending on their preference for these features, even if we recommend the same movie to two different users, the explanations would differ as well. Thus, our explanation of recommendation is personalized, and truly reflects the user's tastes.

3.4.6. User Recommender

The main task of this component is to estimate the similarity between users in terms of their interest in movies using user feature vectors, and to recommend the most similar users to target users. Although the user feature vectors and the movie feature vectors have a similar structure, they are essentially different. The user feature vectors reflect the users' preference for these features, while the movie feature vectors indicate the attributes of the movies. Movie recommendations find those movie vectors in which the attributes satisfy the

target user's preference, by using the inner product of the user feature vector and the movie feature vector, to aggregate the ratings from each feature. But in user recommendations, we care more about the difference between two users' preferences for each feature. If we still use the inner product of two vectors, less weight is given to the features that both users show weak preference for, even though the extent of preference for these features might be very close.

Table 3.3. User-User Partial Interaction Effect

	romantic	sad	astounding	spectacular	scary
$e_{A,f}$	0.10	0.03	0.50	0.40	0.40
$e_{B,f}$	-0.10	0.02	0.50	0.30	0.40
$e_{A,f} \times e_{B,f}$	-0.01	0.0006	0.25	0.12	0.16

For example, in Table 3.3, both users show weak preference for the feature "sad", so the product of the values of "sad" is only 0.0006, which is very small and has little contribution to the overall similarity if we use the inner product. However, since both users show little interest in sad movies, they should be similar in view of this feature. Therefore, we do not simply employ the same logic of movie recommendation. Instead of using Equation 3.1, we apply the cosine similarity, which accounts for the difference between users' preferences on each feature, to estimate the similarity between user feature vectors, and to recommend users with highest similarities to the target user.

As with movie recommendations, we also provide personalized explanations together with the recommended users for target users. When a

target user receives user recommendations, he would care more about what kinds of movies they commonly like (but not dislike). It is reasonable to list the features for which both of them show a strong interest. We rank the features with positive values in both users' vectors according to the partial interaction effect, and select the top *K* features having the highest partial interaction effect as the explanation. For example, in Table 3.3, if user *B* is recommended to user *A*, and only one feature is required, then the feature "astounding" is used as the explanation, since it has positive values in both user *A* and user *B*'s vectors, and has the highest partial interaction effect.

3.5. Experiment and Result

In this section, we first introduce the evaluation metrics used to test the effectiveness of our proposed method. Then we compare our method with rating-based methods and investigate the impact of rating sparsity on different methods. Finally, we compare our method with the tag-based approach for both item recommendation and user recommendation.

3.5.1. Evaluation Metrics

In the experiment, we use the *Mean Absolute Error* (MAE) metric that is commonly used in recommendation research (Herlocker et al. 2004) to evaluate the accuracy of rating predictions of recommendation methods. MAE is defined as:

$$MAE = \frac{\sum_{r_{u,i} \in TestingSet} \left| r_{u,i} - \hat{r}_{u,i} \right|}{\left| TestingSet \right|} .$$

where $r_{u,i}$ is the rating given by user u to item i in the testing dataset; $\hat{r}_{u,i}$ is the predicted rating; and $|TestingSet|$ is the size of the testing dataset.

While accurate prediction is crucial, it does not address one key goal of good recommender systems, which is to cover a wide range of items. Accordingly, we also measure the coverage of item recommendations by using the percentage of items in the testing set of which users' preference can be predicted:

$$ICoverage = \frac{\#Predictable\ Items\ in\ TestingSet}{\left| TestingSet \right|} \times 100\% .$$

In addition to item recommendations, our method also provides user recommendations. To assess the quality of user recommendations, we evaluate the similarity between the recommended users and the target user in terms of interest, and calculate the coverage of recommendations in the user space. Based on the assumption that users with common interests are more likely to tag and rate similar items, the quality of user recommendations can be assessed by measuring the similarity between the set of movies rated and tagged by the recommended users, and the set of movies rated and tagged by the target user. Following Wei et al. (2011), the similarity between two movies is calculated as the average of the cosine similarity of their rating vectors, and the cosine

similarity of their TF-IDF tag term vectors. Given a target user u_t and the top N recommended users RU for user u_t, the similarity of interest between the target user u_t and the recommended users is defined as:

$$InterestSim(u_t) = \frac{\sum_{u_s \in RU} \sum_{i \in I_{u_t}, j \in I_{u_s}} sim(i, j)}{\sum_{u_s \in RU} \left| I_{u_t} \right| \left| I_{u_s} \right|}.$$

where I_{u_t} and I_{u_s} are sets of movies rated and tagged by the target user and recommended user respectively, and *sim(i,j)* is the similarity between movie *i* and *j* from these two sets respectively. We use Interest Similarity as a measure of quality for user recommendation.

Similar to item recommendation, the coverage of user recommendation is referred to as the percentage of users that can be recommended (Shani and Gunawardana 2011):

$$UCoverage = \frac{\#Users\ that\ can\ be\ recommened}{\#Users} \times 100\% .$$

In our experiment, in addition to testing the quality and coverage of our proposed method, we also provide the qualitative results of recommendation explanations.

3.5.2. Experiment Results

3.5.2.1. Comparison with Rating-based Methods

To avoid losing generalizability, we use subsets of three publicly-available

rating datasets from different domains: Movielens in the movie domain, Netflix in the movie & video domain, and BookCrossing in the book domain. Table 3.4 shows the statistics of the rating data used.

Table 3.4. Statistics of Rating Data

Rating Data	Item Count	User Count	Rating Count	Rating Scale	Sparsity Level
MovieLens	1682	943	100000	1 to 5	93.70%
Netflix	1000	4427	56136	1 to 5	98.73%
BookCrossing	1615	1619	35278	1 to 10	98.65%

Table 3.5. Statistics of Review Data

Rating Data (Review Source)	Review Coverage	Word Count per Item	Adjective Count per Item	Unique Adjective Count per Item
MovieLens (IMDb)	98.75%	1679	845	365
Netflix (IMDb)	80.80%	1325	595	276
BookCrossing (GoodReads)	99.94%	3296	1547	552

We also crawl textual user reviews for each item in the rating datasets. Specifically, for Movielens and Netflix items, we obtain user reviews from IMDb, and the source of reviews for BookCrossing items is GoodReads. In order to get high-quality reviews, we rank the reviews according to the number of users who found the review useful in descending order, and select the top 40 reviews for each item. Table 3.5 shows the statistics of the review data.

We split each rating dataset into training set and test set. To further investigate the impact of rating sparsity on recommendations, we vary the

sparsity level of the training data, and compare the prediction accuracy of our proposed method with other state-of-the-art rating-based methods. We introduce a variable *tp* to indicate the percentage of rating data used as a test set. For example, *tp*=20% indicates 20% of the data being used as a test set, and the remaining 80% of the data used as the training set. *Sparsity* of a rating matrix is defined as $1 - \frac{\text{non-zero entires}}{\text{total entries}}$. We vary the value of *tp* to obtain different levels of *sparsity* of the training data. The correlation between *sparsity* and *tp* is shown in Figure 3.3. It is clear that *sparsity* is positively correlated with *tp*.

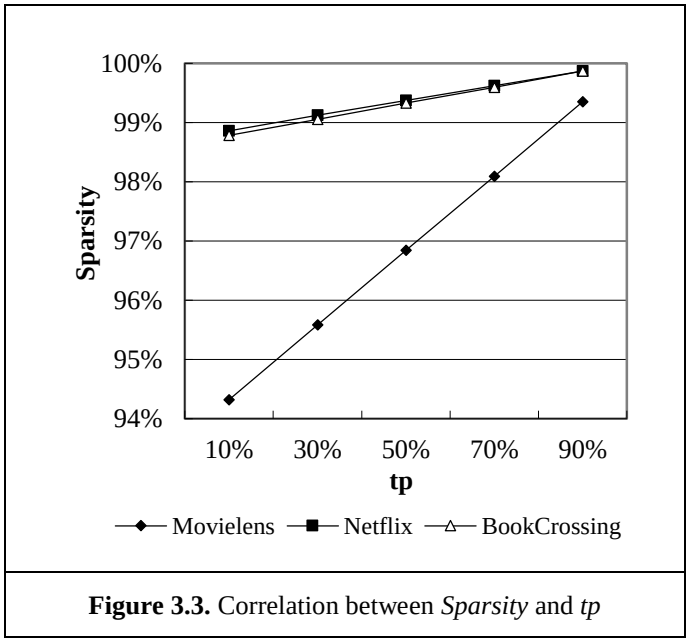


Figure 3.3. Correlation between *Sparsity* and *tp*

We set *tp*=20%, and tune the parameters of our proposed method based on Movielens data. For the SVD-related parameters, we use the same values as reported in (Paterek 2007) that is, λ_2=25, λ_3=10 and λ_4=0.02, since such a configuration also gives the best results in our context. For other parameters, we vary their values to find settings that would give the best results. This occurs

when cluster size M=20, CLF weight λ_1=2 and feature size K=20. We use these particular settings for all the following experiments.

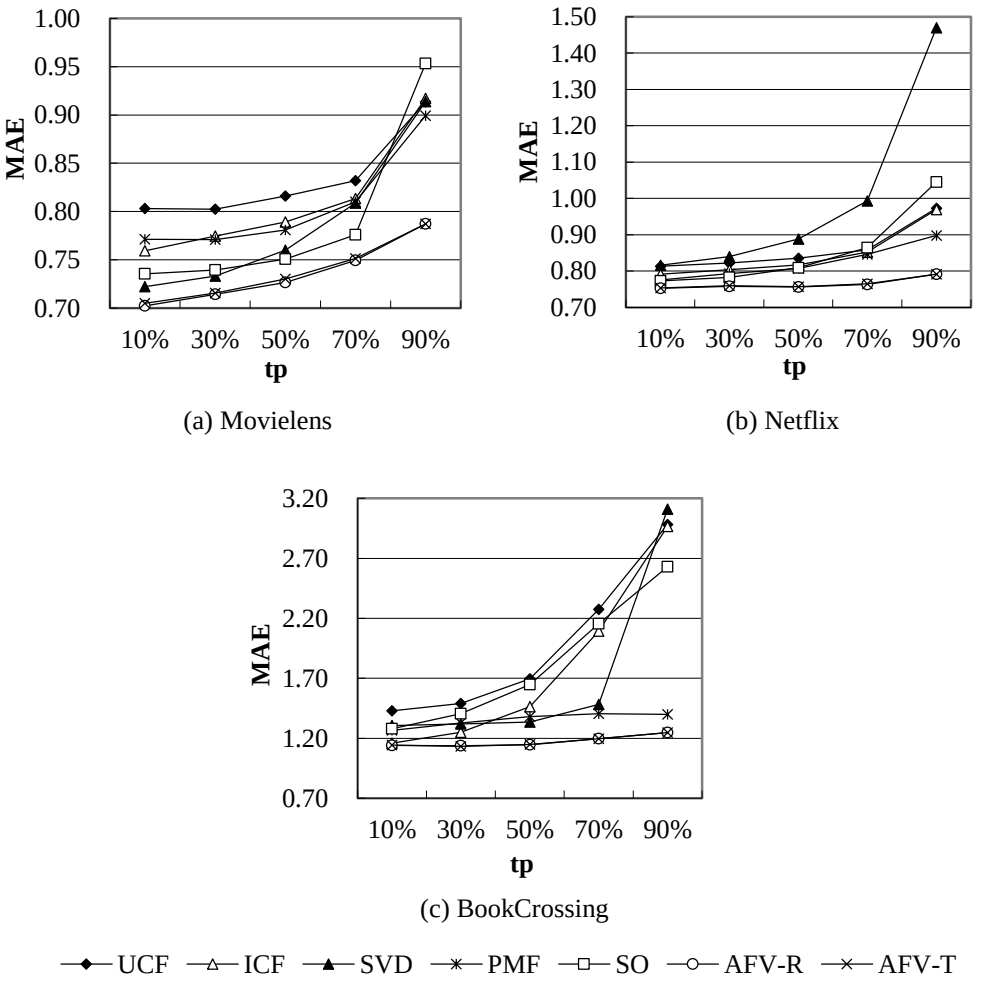


Figure 3.4. Impact of Sparsity

Since we use two approaches to obtain the cluster of similar movies, our method has two variants: *AFV using rating-based clustering* (AFV-R) and *AFV using review-topic-based clustering* (AFV-T). Other rating-based methods for comparison are: User-based CF (UCF) (Resnick et al. 1994), Item-based CF (ICF) (Sarwar et al. 2001), Probabilistic Matrix Factorization (PMF)

(Salakhutdinov and Mnih 2008), Singular Value Decomposition (SVD) (Paterek 2007) and Slope One (SO) (Lemire and Maclachlan 2005). These methods are commonly selected for comparison in recommendation research. Among them, UCF and ICF are the most commonly used techniques in practice, while PMF and SVD represent state-of-the-art rating-based approaches. The results for Movielens, Netflix and BookCrossing are shown in Figure 3.4 (a), (b) and (c) respectively.

From the results, we can see that the prediction errors of our two proposed methods are very close, and they are consistently lower than other methods. The results also show that with the increase of *tp* (or the sparsity level), the prediction errors of all methods increase, but the rate of increase of our methods is slower compared to other methods; that is to say, our methods are less sensitive to data sparsity as compared to other methods. Specifically, at the most sparse settings (i.e. *tp*=90%), our methods reduce prediction errors of the second best method (PMF) by 12.42% on the Movielens dataset, 11.89% on the Netflix dataset and 10.90% on the BookCrossing dataset. The results prove that our methods are effective in alleviating the effect of rating sparsity, and the improvement derived from our method is more salient in extremely sparse settings.

3.5.2.2. Comparison with Tag-based Method

In this experiment, we compare our method with a tag-based method, i.e. Quaternary Sematic Analysis (QSA) (Wei et al. 2011), which represents a state-of-the-art tag-based approach, in both item recommendations and user recommendations.

To compare with the tag-based method, we use the same tag-based dataset as the one used by the QSA method, and compare with the results reported in (Wei et al. 2011). The advantage of comparing with reported results is that the experimental results will not be biased by our own implementation of the existing approaches.

The evaluation dataset is a densely-tagged subset of the Movielens 10M version dataset that consists of 10 million ratings and 95580 tags, applied to 10681 movies by 71567 users. In the original dataset, only 4009 (5.60%) users provided tags for movies, and only 7601 (71.16%) movies received tags from users. In the selected densely-tagged subset, every user gave at least one tag to a movie, and every movie received at least one tag from users. Let <user,movie,tag,rating> denote a tuple; hence the subset comprises 1112 tuples with 201 users, 501 movies and 404 tags. For each item in the densely-tagged subset, we also crawl the top 40 user reviews from IMDb. All results reported below are given by 5-fold cross-validation.

Experiment on Item Recommendation. For item recommendations, we report the results for item rating prediction (i.e. MAE) and item space coverage. In addition to the QSA method, 3 other methods are also included as benchmarks: User-based CF (UCF), Item-based CF (ICF), and Probabilistic Matrix Factorization (PMF).

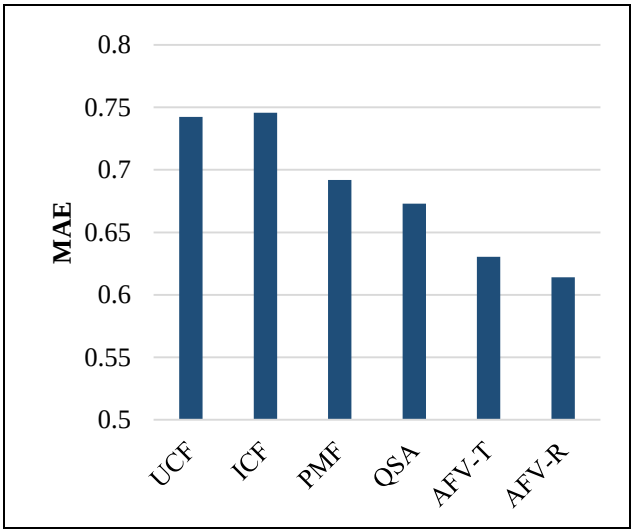


Figure 3.5. Rating Prediction Accuracy
(Lower MAE indicates higher accuracy)

We first compare the performance in item rating predictions of our method with the QSA method. The results shown in Figure 3.5 indicate that both of our AFV-T and AFV-R methods have improved performance over existing methods in terms of accuracy of item rating predictions. Specifically, when compared to traditional CF algorithms, AFV-T reduces prediction errors of UCF and ICF by 15.9% and 15.47% respectively, and AFV-T reduces prediction errors of these two CF algorithms by 17.29% and 17.67% respectively. In comparison with other state-of-the-art methods, AFV-T reduces prediction errors in the rating-

based method (PMF) and tag-based method (QSA) by 8.90% and 6.33% respectively, and AFV-R reduces the prediction errors of these two methods by 11.27% and 8.77% respectively.

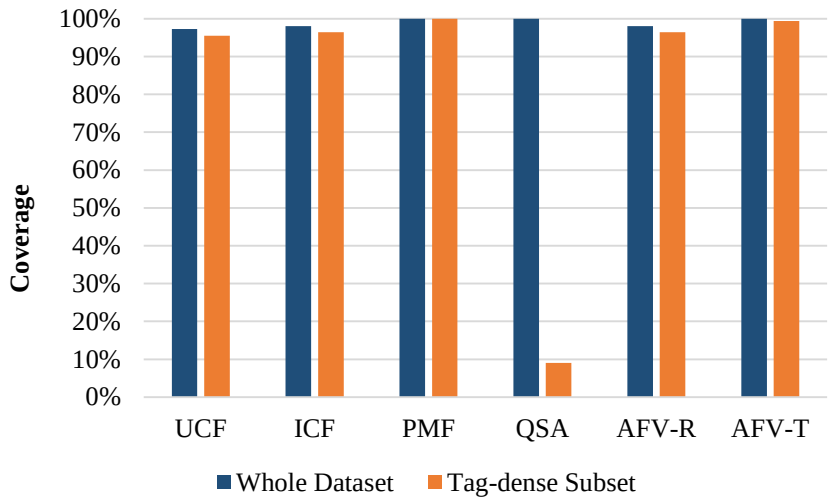


Figure 3.6. Item Space Coverage

In addition to the accuracy of item rating predictions, we also compare the coverage of our methods with other methods. The results for the densely-tagged subset are shown in Figure 3.6. PMF, QSA, and AFV-T achieve 100% coverage, with the two traditional CF approaches also achieving high coverage. Since we apply ICF in AFV-R, the coverage of AFV-R is the same as ICF. Although the QSA method achieves 100% coverage, it is not the case in reality, since it requires every user to provide tags, and every movie to receive tags. In the original dataset, only 5.60% of users provided tags to movies, and only 71.16% of movies received tags from users. We also evaluate the coverage of different methods in the full dataset, and the results in Figure 3.6 show that QSA achieves

an extremely low coverage of only 9.07%. Since the proposed AFV-T and AFV-R methods use external reviews and do not require any tags from the user, they achieve high coverage. Specifically, the coverage of AFV-R is 96.4% (being the same as ICF), and the coverage of AFV-T is independent of user ratings but is determined by the proportion of movies with user reviews, which is 99.4%.

The transparency of the recommender system is often ignored by most CF approaches, whereas our method is able to provide explanations for recommendations given to system users. We will show the qualitative results of recommendation explanations using our method.

Table 3.6. Explanation for Item Recommendation

	My Neighbor Totoro	*Grave of the Fireflies*
User *A*	curious, suitable, imaginative, warm, magical, friendly, sentimental, sweet	magical, suitable, cold, gorgeous, sentimental, beautiful, happy, extraordinary
User *B*	endearing, suitable, imaginative, giant, cute, poetic, boundless, fantastical	giant, astonished, live, engrossing, animated, suitable, poetic, gentle
User *C*	lovely, delightful, happy, gentle, spectacular, engaged, curious, magical	lovely, gentle, happy, magical, afraid, heartfelt, cold, engrossing

Applying the proposed AFV-R method, which has the highest accuracy in predictions, we recommend 5 movies to each user. We arbitrarily select three users, who have 2 movies in common in their recommendation list, to illustrate the qualitative results of recommendation explanations given by our method. The two movies in common are *My Neighbor Totoro* and *Grave of the Fireflies*.

Table 3.6 shows the 8 listed features for each movie as explanations for recommending the movie to each user.

As shown by the results, the explanation for recommending the same movie is personalized for different users, taking movie features and users' tastes on each feature into consideration. For example, we recommend the movie *Grave of the Fireflies* to user *A*, together with the explanation: "magical, suitable, cold, *gorgeous*, sentimental, beautiful and happy and extraordinary"; while the explanation for the same movie recommended to user *B* is: "giant, astonished, live, engrossing, animated, suitable, poetic and gentle". In addition to providing the recommendation for a movie, the explanation gives the users more insight into the recommendation mechanism, therefore making the recommendation more trustworthy and acceptable.

Experiment on User Recommendation. To evaluate the effectiveness of our proposed method in user recommendations, we compare our AFV-R and AFV-T methods in terms of the average similarity of interest between target users and recommended users, as well as user space coverage, with the QSA method, representing a state-of-the-art tag-based user RS. We also include a random method (RAN) which randomly selects *N* users for recommendation, as a baseline method.

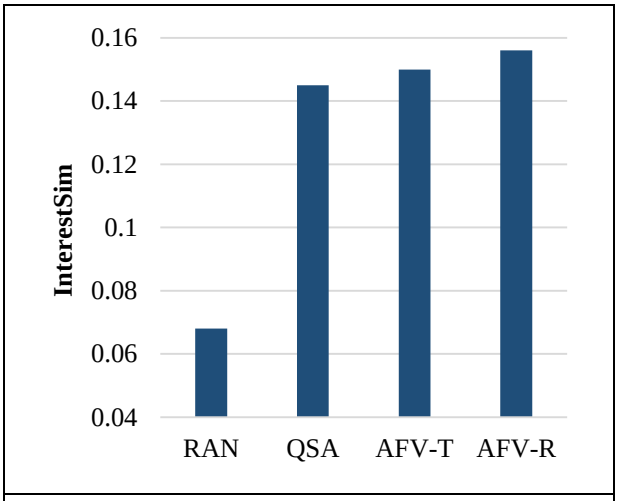


Figure 3.7. Average Interest Similarity
(Higher InterestSim indicates higher accuracy)

In this experiment, we recommend 3 users for every user in the tag-densely-tagged dataset. Figure 3.7 shows the average similarity of interest between target users and recommended users, using different methods. From the results, we can see that both the QSA method and our AFV methods increase the similarity of interest between target users and recommended users. Furthermore, our AFV methods improves on the QSA method in terms of the similarity in interest between target users and recommended users. Specifically, the interest similarity values of AFV-T and AFV-R are 0.150 and 0.156 respectively, whereas this value is 0.145 using QSA. That is, AFV-T and AFV-R increase the interest similarity of QSA by 3.45% and 7.14% respectively. The results show that our proposed method is effective in selecting similar users and outperforms state-of-the-art tag-based method.

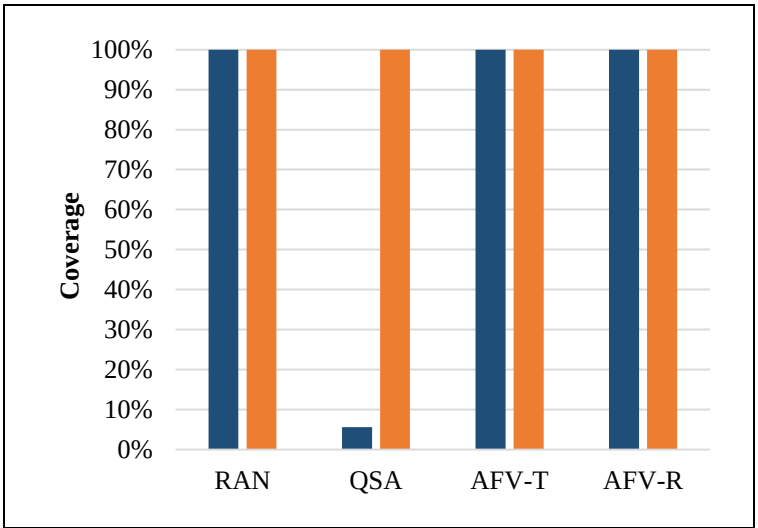


Figure 3.8. User Space Coverage

Besides interest similarity, we also evaluate the effectiveness of our methods by comparing the coverage of recommendations in the user space. Similar to item recommendations, we first evaluate the coverage using the densely-tagged subset, and all methods achieve 100% coverage, since every user gave tags to movies, and every movie received tags from users in this subset. We then repeat the experiment using the full dataset, where the majority of users did not provide tags. From the results in Figure 3.8, we can see that the tag-based QSA method achieves an extremely low coverage of only 5.60%, which is the same as the proportion of users who provided tags in the dataset; while our AFV methods, which use external user reviews and do not rely on tags, still achieve 100% coverage. The results indicate that our proposed AFV methods significantly outperform the state-of-the-art tag-based method (i.e. QSA) in user space coverage.

Our proposed methods are able to provide explanations for not only item

recommendations, but also user recommendations. To illustrate the quality of the explanations for user recommendations, we apply the AFV-R method and recommend 5 users to every system user. For every recommended user, we provide 8 features indicating similar taste aspects to target users as the explanation. We arbitrarily select two users, and list the explanation for the two recommended users they receive in Table 3.7.

Table 3.7. Explanation for User Recommendation

	Recommended User 1	Recommended User 2
User D	comic, colorful, heroic, positive, controversial, theatrical, suspicious, gothic	vocal, ludicrous, legendary, gorgeous, terrible, musical, colorful, creepy
User E	romantic, lovely, promising, emotional, comic, social, amusing, conventional	smart, bright, fresh, ridiculous, tremendous, stylish, political, complex

3.6. Conclusion

In this work, we show that adjective features embedded in user reviews are useful for characterizing movie features as well as users' tastes, and can be employed by recommendation techniques to address sparsity and transparency issues. We employ POS tagging and propose introducing Cluster Frequency (CLF) into the traditional TF-IDF term weighting scheme, to extract adjective features from external user reviews, highlighting terms that help discriminate between implicit item aspects, and balancing the representativeness and generalizability of the extracted features. We also incorporate the extracted

adjective features into a specific recommendation technique, i.e. Singular Value Decomposition (SVD), to illustrate the effectiveness of using adjective features. The experiment results show that the proposed AFV method makes a significant difference to the quality of the state-of-the-art rating-based method (i.e. reducing 12.42% prediction errors of PMF) in settings where ratings are extremely sparse, and outperforms state-of-the-art methods in item recommendations and user recommendations, in terms of both quality and coverage. Specifically, in item recommendations, our AFV method reduces the prediction errors of the state-of-the-art tag-based method by 11.27%, and in user recommendation, it increases the interest similarity of the state-of-the-art tag-based method by 7.14%. Moreover, our AFV method always achieves high coverage of both item and user recommendations, while the coverage of tag-based methods is extremely low when tags are sparse, which is always the case in reality. In addition to recommending items and users, the AFV method is also able to provide personalized explanations for recommendations to users, increasing trust in the recommendation.

There are some limitations to our work. Firstly, we only considered the adjective features and ignored other descriptive attributes of items. Our method can be extended to incorporate other descriptive attributes, which may generate more accurate recommendations and higher-quality explanations. Secondly, we did not consider the semantic relationship between adjective features, which is

a potential direction for future work.

Although our recommendation architecture was evaluated on single domains, it can easily be applied to cross domains. Since the extracted adjective features capture user tastes on a higher and more abstract level, it will be interesting to evaluate the application of our method in cross-domain recommendation in future work.

CHAPTER 4. STUDY ON USING CRITIC REVIEWS TO BOOST NEW ITEM RECOMMENDATION

4.1. Introduction

Facilitated by the rapid development of technology, the barriers of entry for production of new items have lowered considerably. As a consequence, in most domains of consumer products studied, new items are being added regularly at a speed never seen before. For example, according to (Datta et al. 2012), 100 new movies, 250 new books and up to 15,000 new mobile apps are released per week on average. The huge number of new items can hardly be accessed by consumers without a mechanism that effectively supports the discovery of new items.

In a recent development, RS has shown promise to help consumers make good choices amidst an overwhelming number of alternative items, by providing personalized recommendations. However, as illustrated in the previous chapters, existing recommendation techniques suffer from data sparsity.

Collaborative Filtering (CF) works only if the items are already well-known (i.e. the items have been previously purchased or rated by many users), but it lacks the ability to discover and recommend new items since the user

ratings required by CF are extremely sparse, or totally unavailable, in the case of new items. The problem occurs when new items are continuously added but are unable to be recommended. This problem is also known as the *new item problem* or *cold start item problem* that has been identified as a major challenge of RS (Schein et al. 2002). An intuitive solution to the new item problem is to adopt Content-Based (CB) approaches that typically match user preference data with item attribute information, to help bridge the gap between existing and new items. However, such methods encounter the limitation of insufficient item attributes. Research has found that the limited number of descriptive attributes assigned to items is insufficient to determine distinguishing features of items, which might be necessary for the elicitation of users' taste aspects (Lops et al. 2011). For example, in the movie RS, a user may prefer dramas about school life but dislike dramas with racial discrimination. If genre is used as an indicator of users' preferences, it will fail to differentiate between these two detailed aspects of user tastes within a single genre. A possible approach to address this limitation is to incorporate other item information into the RS, to represent item features and define user taste aspects.

We started exploring external data that can possibly be incorporated, and noticed that when people were choosing a digital product to buy, a book to read or a movie to watch etc., they would first search for online review articles about these items, and then evaluate them based on their features described in such

articles. It motivated us to consider automating this process by incorporating external review articles in the RS. On one hand, online review articles are available in abundance, even for new items. In the movie domain, for example, we analyzed two famous online movie review aggregators, i.e. IMDb and Rotten Tomatoes, and found that on both platforms, 92% of new movies[3] (movies still playing in theaters) have critic-reviewed articles. The average numbers of critic-reviewed articles per new movie on both platforms are 69 and 17 respectively. However, if we use Wikipedia, as proposed by Katz et al. (2011), only 65% of new movies have corresponding content pages. Clearly, review articles have a dominant advantage in quantity and the coverage of new items, which enables us to address the new item problem with substantial supplementary information. On the other hand, compared to descriptive attributes, review articles cover more item features. For example, in critic reviews of the movie *The Graduate*, we are able to infer that the topic of this movie is about youth and love, as well as many other features unable to be captured by general descriptive attributes. In short, the nature of review articles makes it an ideal source of supplementary data for recommendation. In this study, we will address the new item problem of RS by incorporating online review articles.

[3] Since IMDb and Rotten Tomatoes are English-oriented platforms, we only consider English movies.

Although online review articles show promise, from a technical perspective, there are two challenges to incorporating such data in the recommendation process. First, review articles are unstructured free-text. A proper text model is required to quantify the textural contents. Traditional RS dealing with textural contents usually represent item features and users' taste aspects at the word level (Ahn et al. 2007; Katz et al. 2011; Spaeth and Desmarais 2013), which may result in the problem of over-specification. For example, a user may prefer family movies, in which the word "mother" may appear frequently in their textual descriptions, but this does not mean that this user must like all the movies whose textual descriptions contain "mother". To address this problem, we propose to use an advanced topic modeling approach that models review articles at the topic level and represents items with topic distributions. Second, it is crucial to effectively integrate item features represented by topic distributions and user ratings in recommendation. We adapt Non-negative Matrix Factorization (NMF) to fully utilize the user ratings and item features, which would be helpful in improving the recommendation quality.

We use the topics of the critic reviews from existing items to define the taste aspects of the users, and utilize the user ratings to estimate the extent to which a user likes a particular topic. When new items are added, we collect their critic reviews, and infer their topic distributions. Then the new item problem can be alleviated by matching the users' topic preference with the topic

distributions of the new items.

The results of our experiment conducted in a real world data set show that our method is efficient and can not only generate high quality new item recommendations in cold start settings which are not supported by many state-of-the-art methods, but also outperform the state-of-the-art methods when recommending existing items especially in rating-sparse settings. Specifically, our method reduces the prediction errors of the state-of-the-art method using item typology based on item keywords by 5.78% and improves the ranking accuracy of the state-of-the-art method by 12.91% in rating-sparse settings.

The rest of the sections in this chapter are organized as follows. First we introduce the background to the research and review the related work. Then we present our proposed recommendation architecture including the intuition and the detail description of each component. The remainder of this chapter then presents the experiment and results, and finally, we conclude by summarizing this study.

4.2. Related Work

Our method is a Content-Based (CB) approach using the external critic reviews of items to address the cold start problem. We employ an advanced topic modeling approach, Partially Labeled Dirichlet Allocation (PLDA), to represent the critic reviews at the topic level. We also adapt Non-negative Matrix

Factorization (NMF) to fully utilize the rating and content information. The related work will be introduced in the following.

4.2.1. Partially Labeled Dirichlet Allocation

Latent Dirichlet Allocation (LDA) (Blei et al. 2003) is a generative probabilistic model that applies hierarchical Bayesian analysis to discover the semantic structure in a text corpus. The basic idea of LDA is to represent a document as a multinomial distribution over latent topics, each of which is characterized by a distribution over words. LDA is an unsupervised learning model. The generated unsupervised topics are powerful for exploring the underlying sub-structure, but it may be difficult to interpret their meaning and they usually do not align with human provided labels. Labeled LDA (Ramage et al. 2009) is a supervised extension of LDA that requires the topics to align with the pre-defined labels assigned to the documents, but it may fail to capture the broad patterns in the corpus.

In a recent development, a semi-supervised model, i.e. Partially Labeled Dirichlet Allocation (PLDA) (Ramage et al. 2011), has been proposed. PLDA takes full advantage of both supervised and unsupervised approaches. It is able to discover any number of hidden topics under each pre-defined label, and it also has the ability to explore the latent topics across the whole corpus.

There are a few existing works applying LDA in recommendation. For

example, a recent study (Cai et al. 2014) proposed a TyCo method which uses LDA to model keywords of movies and then construct item typicality for further recommendation. But no reported work using PLDA in recommendation has been found. The nature of PLDA makes it suitable for our purpose of uncovering topics in the critic reviews. The learned topics are then incorporated with user ratings by applying NMF in our method.

4.2.2. Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) (Lee and Seung 1999) is a powerful dimension reduction tool for non-negative data and has been successfully adopted in many fields such as signal processing and text mining. Given a non-negative matrix $V \in \mathbb{R}_{+}^{m,n}$ and a specified positive integer $k < \min(m,n)$, NMF seeks two non-negative matrices $W \in \mathbb{R}_{+}^{m,k}$ and $H \in \mathbb{R}_{+}^{k,n}$ so that their product WH approximates the original matrix V. The intuition of NMF is to use a linear combination of the basis vectors (i.e. the rows in W) and the coefficient vectors (i.e. the columns in H) to approximate the input vectors (i.e. the rows in V). NMF can be solved as a problem of minimizing the error function, which is typically the square error or Kullback-Leibler divergence, and coordinate descent algorithms (Hsieh and Dhillon 2011; Seung and Lee 2001) are commonly used. In our method, we adapt NMF for our context by redefining the error function and using a simple and effective projected gradient descent approach (Lin 2007) to solve the optimization problem.

4.3. Intuition and Overview

Similar to study one, although our proposed method is generally applicable for any domain of consumer products, we make it easier to explain our idea and to compare our work with existing ones by choosing a specific example domain. As movies are the most studied consumer domain in recommendation research, we present our work by using the domain of movies. That is to say, from this point forward, we will present our method as a technique of providing movie recommendations to users.

The objective of our method is to predict the users' preference for movies which are unknown to them, and to recommend movies with the highest predicted ratings to them. In order to predict the target user's preference for a given movie, we need to know what kinds of movies he has liked in the past, and what kinds of movie the given movie belongs to. A common way to do this is to use the descriptive attributes of the movies, such as the genre and the director, to define the characteristics of the movie and the users' taste aspects. For example, if this user has highly rated scientific movies directed by Spielberg, and the given movie happens to be scientific and directed by Spielberg, then the predicted rating would be higher.

However, a user's taste may be far beyond the aspects defined by the limited number of descriptive attributes. For example, a user may prefer

comedies about school life or dramas about racial discrimination. This leads us to contemplate whether there are other types of data that can be incorporated to cover more features of the movies to capture more aspects of users' taste. We notice that a specific kind of movie information, i.e. critic reviews, can be found in many online systems like Rotten Tomatoes[4], which is widely known as a movie review aggregator. Unlike other user generated content or user preference data that only can be found long after the movie is released to the public, critic review articles are available in abundance even before the release of the movie. For example, 22 high quality critic review articles of a recent movie *Mud* can be found in Rotten Tomatoes even one week before its release. The availability of expert critic reviews fulfills our requirement for information on new movies, and their contents may cover all possible aspects of the movies. Therefore, we incorporate external expert critic reviews to define movie features.

Since expert critic reviews are presented in the form of free text, a proper text model should be used to represent movies with these text contents. We apply PLDA to the expert critic reviews to infer the topics of movies under their genres, as well as the topics that are shared by all the genres. For example, a topic under the genre "drama" may be related to "racial discrimination", and a general topic may be related to "family", since many movies in different genres

[4] http://www.rottentomatoes.com

may involve talking about family related matters. The adaption of Non-negative Matrix Factorization (NMF) allows us to calculate users' preference for each topic based on their rating data. Therefore, rating prediction becomes a problem of estimating to what extent the movie topic distribution matches the user topic preference. Given a new movie with a collection of expert critic reviews, we are able to tell to what extent this movie is associated with which topics, and new item recommendation can be performed by matching up the movie topic distribution with the user topic preference.

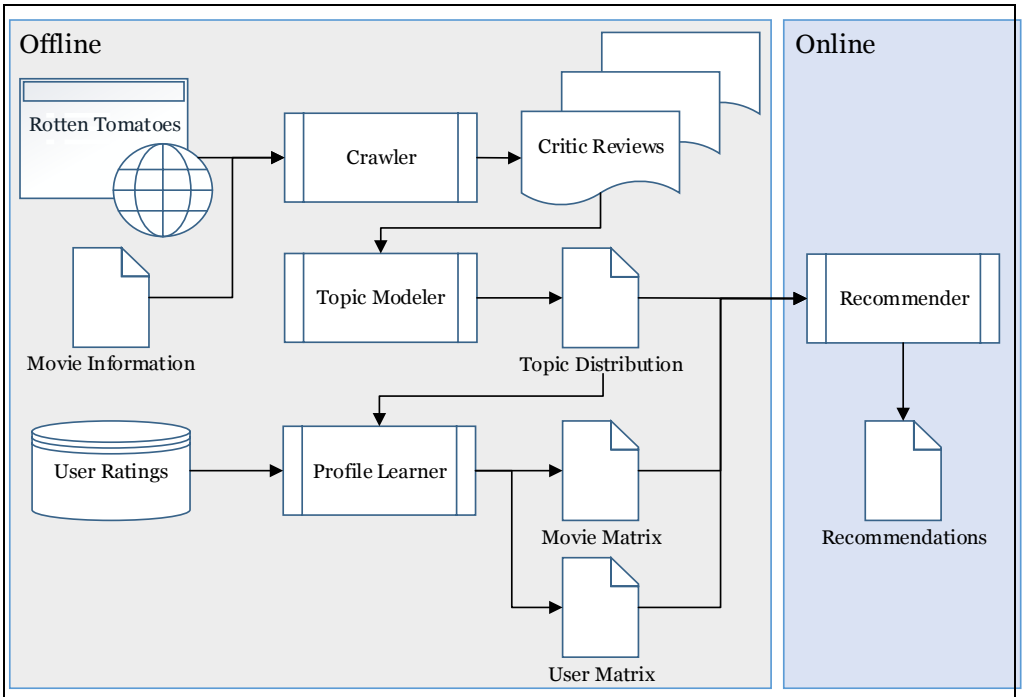


Figure 4.1. Proposed Architecture

4.4. Solution Details

The overview of our movie recommendation architecture is shown in Figure 4.1. We have designed and implemented four components in the architecture to

realize our recommendation engine. The four components are: *Crawler*, *Topic Modeler*, *Profile Learner* and *Recommender*. Among these components, Recommender is the only one running online, while the other three can run offline. Specifically, for existing movies, we use the Crawler to collect critic reviews from external websites. The contents of critic reviews are then analyzed by the Topic Modeler to uncover the underlying topics in the movies. The Profile Learner utilizes user ratings and the movie topic distributions to learn about the users' preferences. Then, the Recommender generates personalized movie recommendations by matching up the user preference with the movie features. For new movies, their topic distribution is inferred from the topic model trained by the existing movies, and then are used together with the user profiles learned from existing movies to generate recommendations. The PLDA employed in the Topic Modeler and the adaption of NMF in the Profile Learner distinguishes our proposed method from other methods, which also contributes to generating more efficient and higher quality recommendations even with cold start settings. More details of each component will be introduced in the ensuing sections of the chapter.

4.4.1. Crawler

The main task of the Crawler is to collect critic reviews from external websites via Rotten Tomatoes. Rotten Tomatoes aggregates critic reviews from reliable sources with good reputation and compiles a list of their URLs. It also allows

users to select reviews from top critics. Using the titles and release years of movies to match the movie information in Rotten Tomatoes via its search API, we obtain a list of critic review URLs. For each movie, we crawl 20 webpages of critic reviews. To ensure quality, we primarily use reviews from top critics. If their number is less than 20, we also use reviews from other critics. Since the reviews are from different websites, the structures of the webpages containing the review contents are different. We need to use a content extractor (Kohlschütter et al. 2010) to extract the review contents from these webpages. We filter out reviews that are not written in English and those that are too short (less than 100 words). The extracted review contents are then passed to the Topic Modeler.

4.4.2. Topic Modeler

We use the Topic Modeler to represent the movies at the topic level, and a topic is a multinomial distribution over words. The Topic Modeler works by learning and inferring the topic distribution of movies from their critic reviews. We employ PLDA that allows us to use the well-structured attributes (i.e. genre, director, actor, etc.) of movies to supervise the topic learning process, which contributes to higher quality and more interpretable learned topics. The attribute we choose is genre, since genre has been proven to be a good indicator of users' taste (Manzato 2012). PLDA regards genre as a high level category of movies, and learns the specified number of latent topics under each genre. It also

uncovers the global shared background topics that may not belong to a specific genre. Table 4.1 shows some example topics automatically learned from the critic reviews. Each topic is represented as a set of most common words in this topic. As we can see, two global shared topics can be interpreted as "family" and "life" respectively, which means that movies in different genres may talk about the same topics. Global shared topics are important since they capture broad patterns across the whole corpus of critic reviews.

Table 4.1. Examples of Topics	
(Global)	
Topic 1	family, daughter, young, wife, marry, adaptation, century, miss, father, country, son, base
Topic 2	young, sex, girl, feel, image, life, sense, begin, leave, relationship, death, child
Drama	
Topic 1	black, american, young, stone, drug, white, justice, president, kill, murder, violence, war
Topic 2	student, school, white, black, teacher, young, priest, town, south, class, dean, singleton
Comedy	
Topic 1	player, funny, stern, fashion, altman, game, big, fan, wife, team, call, jake
Topic 2	girl, gay, school, dance, lane, sex, drag, high, queen, goldberg, student, funny

Specifically, we use G to denote a set of genres and G_i ($1 \le i \le |G|$) indicates the i th genre. For each genre G_i, we assign some number of topics T_{G_i} to it, where each topic $T_{G_i,j}$ ($1 \le j \le |T_{G_i}|$) is a representation of a multinomial distribution over all words in the vocabulary of the critic reviews. The number of topics for each genre can be different, which allows us to assign more topics

to those genres having a higher proportion of movies. In order to explore the global shared topics beyond the genres, a special label is used which can be interpreted as the "global" genre that is shared by all the movies, and some number of latent topics T_{global} are also assigned to it. PLDA is a generative model assuming that each word *w* in the critic reviews of a movie *m* belonging to a set of genres Λ_m are generated as follows: first, a genre *g* in Λ_m is drawn from a multinomial distribution of size $|\Lambda_m|$, then a topic *t* in T_g is drawn from a multinomial distribution of size $|T_g|$, and the word *w* is drawn from a multinomial distribution over the whole vocabulary in this topic. Intuitively, the probability that a word in the critic reviews of a movie is picked is in proportion to the aggregation of the following probabilities: (1) how likely this movie belongs to the genre *g*; (2) how likely genre *g* belongs to the topic *t*; and (3) how likely topic *t* has this word. Details of the algorithm for learning and inferring the model parameters can be found in (Asuncion et al. 2009).

We can use the critic reviews from a subset of the existing movies to build the topic model by learning the topic distribution. When a new movie is added, its critic reviews can be used to infer its topic distribution based on the learned topic model. The output of the Topic Modeler is matrix *P* representing the topic distribution of the movies. Each column of *P* is a vector $\overrightarrow{P_m^T}$ that represents the multinomial distribution over all topics for a movie *m*, and each element $P_{t,m}$ in this vector is the probability that movie *m* belongs to topic *t*. All elements in $\overrightarrow{P_m^T}$

sum up to 1, that is, for all *m*,

$$\sum_t P_{t,m} = 1.$$

Table 4.2 shows examples of distributions over the above-mentioned example topics for 3 movies. *American History X* is a drama, *Van Wilder* is a comedy and *The Graduate* is both drama and comedy.

Table 4.2. Examples of Movie Topic Distribution

	(Global) 1	(Global) 2	Drama 1	Drama 2	Comedy 1	Comedy 2
American History X	0.2	0.098	0.002	0.7	0	0
Van Wilder	0.18	0.2	0	0	0.02	0.6
The Graduate	0.103	0.116	0.001	0.4	0.3	0.08

4.4.3. Profile Learner

The Profile Learner is a core component in the recommendation engine. With the topic distribution of movies, Profile Learner utilizes the user ratings to learn user preferences by computing to what extent a given user likes a particular topic. Specifically, in order to isolate the users' topic preference from other factors, we divide a user rating given to a movie into 4 parts: basis rating (i.e. overall average), user bias (i.e. some users may tend to rate higher or lower than other users), movie bias (i.e. some movies may tend to receive higher or lower ratings than other movies), and user topic preference. The original rating matrix *X* is approximated by:

$$X \approx S + B_U + B_I + UI .$$

where each element $X_{i,j}$ in the matrix X is the rating given by user i to movie j; all elements in S are equal to the global average rating μ; all elements in the i th row of matrix B_U have the same value that is equal to the user rating bias $ubias_i$, and all elements in the j th column of matrix B_I have the same value that is equal to the movie rating bias $mbias_j$. $U \in \mathbb{R}_+^{m,k}$ and $I \in \mathbb{R}_+^{k,n}$, where m is the total number of users, n is the total number of movies, and k is the total number of topics. $U_{i,t}$ indicates the extent to which user i prefers topic t, and $I_{t,j}$ indicates the extent to which movie j belongs to topic t.

By adapting NMF, the Profile Learner decomposes the original user-rating matrix into two matrices U and I to represent users' topic preferences and movies' topics respectively. The decomposed matrices should satisfy two criteria: (a) the product of the user matrix U and movie matrix I should approximate the original matrix after adding the basis rating and rating bias; (b) the normalized movie matrix $\tilde{I}$ should approximate the topic distribution matrix P. The movie matrix I acts a bridge between two types of data, i.e. user ratings and movie critic reviews, by satisfying the above-mentioned criteria. The first criterion can be satisfied by solving the least square error problem, and the second criterion can be satisfied by minimizing the Kullaback-Leibler (KL) divergence (Kullback 1987) between the normalized movie matrix $\tilde{I}$ and the topic distribution matrix P. Since the elements in the column of P sum up to 1, we make a column-wise normalization for movie matrix I, that is, for all j:

$$\tilde{I}_{t,j} = \frac{I_{t,j}}{\sum_{t=1}^{k} I_{t,j}}.$$

According to the definition, the KL divergence between *P and* $\tilde{I}$ is:

$$D_{KL}(P \| \tilde{I}) = \sum_{t=1}^{k}\sum_{j=1}^{n} P_{t,j} \log \frac{P_{t,j}}{\tilde{I}_{t,j}} = \sum_{t=1}^{k}\sum_{j=1}^{n} P_{t,j} \log \frac{P_{t,j}}{I_{t,j}} + \log \sum_{t=1}^{k}\sum_{j=1}^{n} I_{t,j}.$$

Then the two criteria can be satisfied by solving the objective below:

$$\min_{B_U, B_I, U, I} f(B_U, B_I, U, I) =$$
$$\sum_{i=1}^{m}\sum_{j=1}^{n}\{[X_{i,j} - S_{i,j} - B_{Ui,j} - B_{Ii,j} - (UI)_{i,j}]^2 + \lambda_1[{B_{Ui,j}}^2 + {B_{Ii,j}}^2 + \sum_{t=1}^{k}({U_{i,t}}^2 + {I_{t,j}}^2)]\} + \lambda_2 D_{KL}(P \| \tilde{I}).$$
$$\text{subject to } U_{i,t} > 0, I_{t,j} > 0, \forall i, j, t.$$

where λ_1 indicates the extent of penalizing the magnitudes of the parameters to avoid over fitting; and λ_2 indicates the weight given to the topic distribution of critic reviews.

The values of the elements in *U* and *I* are initiated by assigning a random value *s* (0<*s*<0.1) that follows a Gaussian distribution. The movie rating bias and user rating bias are initiated as the average deviation from the global average rating μ with regularization parameters λ_3 and λ_4 as follows:

$$mbias_j = \frac{\sum_{i=1}^{m}(X_{i,j} - \mu),\ if\ X_{i,j} \neq 0}{\lambda_3 + \#\text{non-zero elements in } \overline{X_j^T}}.$$

$$ubias_i = \frac{\sum_{j=1}^{n}(X_{i,j} - \mu - mbias_j),\ if\ X_{i,j} \neq 0}{\lambda_4 + \#\text{non-zero elements in } \overline{X_i}}.$$

To satisfy the non-negative constraint, we employ a project gradient method to update the parameters. Details of the algorithm can be found in (Lin

2007).

4.4.4. Recommender

The Recommender is the only component running online, while the other components can run offline. The objective of the Recommender is to match the user preference to the movie features in terms of topics and to generate movie recommendations for the users efficiently. For the existing movies, we can predict the users' ratings by using the approximation: $S + B_U + B_I + UI$. For the new movies, we don't have the item matrix I or the movie rating bias B_I since no user rating is available for them so that we cannot predict the users' real ratings, but we can still estimate the users' preference by using the topic matrix P instead of I. However, the scale of the predicted ratings given by UP for new movies is different from that given by $S + B_U + B_I + UI$ for existing movies. In order to unify the scale of predicted ratings and to make the existing and new items comparable, we predict another rating for each existing movie using the product of the user matrix U and the normalized item matrix $\tilde{I}$, and recommendations are generated by selecting the items having the highest predicted ratings given by UP (for new movies) and $U\tilde{I}$ (for existing movies).

For example, if we want to predict the rating of movie j for user i, and the corresponding row or column in the matrix P, I, $\tilde{I}$ and U are shown in Table 4.3, assuming that the overall average rating μ=2.5, user i tends to rate 0.5 higher

than other users, i.e. $ubias_i$=0.5 and movie *j* tends receive ratings that are 0.2 lower than other movies, i.e. $mbias_j$=−0.2, then the user's rating on this movie is predicted by: $\mu + ubias_i + mbias_j + \overrightarrow{U_i} \cdot \overrightarrow{I_j^T}$ =2.95. In order to make movie *j* comparable with the new movies, another predicted rating is given by: $\overrightarrow{U_i} \cdot \overrightarrow{\tilde{I}_j^T}$ =0.313. Suppose that movie *j* is a new movie and we don't have the matrix *I* and $mbias_j$, the predicted rating is given by $\overrightarrow{U_i} \cdot \overrightarrow{P_j^T}$ =0.318.

Table 4.3. Example of Vectors

	Latent 1	Latent 2	Drama 1	Drama 2	Comedy 1	Comedy 2
$\overrightarrow{P_j^T}$	0.180	0.200	0.000	0.000	0.020	0.600
$\overrightarrow{I_j^T}$	0.100	0.100	0.002	0.001	0.010	0.280
$\overrightarrow{\tilde{I}_j^T}$	0.203	0.203	0.004	0.002	0.020	0.568
$\overrightarrow{U_i}$	0.300	0.100	0.150	0.020	0.200	0.400

Since the dimension of the original rating matrix is reduced, the rating prediction process in this online component can be efficient.

4.5. Experiment and Results

In this section, we describe the experiment and the results to show the effectiveness of our proposed method. We start from the evaluation metrics, and then proceed to introduce the data set used, the configuration and the environment of the experiment, followed by the results of the experiment. We show the impact of data sparsity, test the efficiency, compare the prediction errors and ranking accuracy with state-of-the-art methods in recommending

existing items, and evaluate the recommendation quality in new item recommendation.

4.5.1. Evaluation Metrics

The accuracy of rating prediction is the most discussed property in recommendation research. Most research in recommender systems relies on the basic assumption that a recommender system providing "accurate predictions" would be preferred by users (Shani and Gunawardana 2011), and seeks algorithms that provide more accurate rating predictions. In line with this, we choose a commonly used metric in recommendation research, i.e. Mean Absolute Error (MAE) (Herlocker et al. 2004), to evaluate the accuracy of rating prediction. MAE is defined as:

$$MAE = \frac{\sum_{r_{u,i} \in TestingSet} \left| r_{u,i} - \hat{r}_{u,i} \right|}{\left| TestingSet \right|} .$$

where $r_{u,i}$ is the rating given by user u to item i in the testing dataset; $\hat{r}_{u,i}$ is the predicted rating; and $|TestingSet|$ is the size of testing dataset.

Although accurate prediction is crucial, in most cases, the recommendations are presented to the users as a list of items, and the order of items in the list is also important. Some research found that accurate prediction does not guarantee the correct order of the recommendations (McNee et al. 2006). A good RS should not only provide accurate rating predictions, but also

should rank the recommended items correctly. In our experiment, we use the NDCG@*k* *(Järvelin and Kekäläinen 2002)* that is also a commonly used metric in recommendation to measure the ranking accuracy. NDCG@*k* is defined as:

$$NDCG@k = \frac{1}{|U|}\sum_{u \in U} Z_u \sum_{p=1}^{k} \frac{2^{r_{u,p}} - 1}{\log(1+p)}.$$

where U is the set of users; Z_u is a normalization factor to guarantee that for the perfect ranking, the NDCG value is 1; p is the position of the recommended item in the list; and $r_{u,p}$ is the rating given by the user u to the item at position p.

4.5.2. Experiment Setup

In order to compare our method with other methods in the experiment, we use the MovieLens dataset that is publicly available and is widely used in other research. The dataset consists of 100,000 ratings given by 943 users to 1682 movies. The user ratings are on a scale of 1 to 5, with 1 being bad and 5 being excellent. The percentage of missing ratings in the dataset (aka. sparsity level) is $(1-\frac{100000}{1682\times 943})\times 100\% = 93.69\%$. The dataset also provides some movie information such as the title, release year and genre. We use the title and release year to get the URLs of critic reviews via the API provided by Rotten Tomatoes, and crawl the corresponding critic reviews from external websites. 98.81% of the movies in the dataset have critic reviews in Rotten Tomatoes.

Before conducting the experiment, we need to assign the number of topics

to each genre, and determine the values of some parameters in the Profile Learner. We assign 4 topics as the global shared latent topics, and for other genres, the number of topics is in proportion to the number of movies in this genre. For example, we assign 4 topics to the genre "children" that has 119 movies, and assign 2 topics to the genre "musical" that has 56 movies. There are 17 genres and the total number of topics is 68. We vary the parameters to find the settings giving the best results. This occurs when $\lambda_1 = 0.02$, $\lambda_2 = 0.5$, $\lambda_3 = 25$ and $\lambda_4 = 10$. We use the same configuration in all the following experiments.

All experiments are conducted using a PC with Intel Core™2 Quad Processor Q9300 CPU (2.50 GHz), 4GB RAM, Windows 7 Professional Operating System and J2SE 7 platform.

4.5.3. Experimental Results

In the experiment, we implement two most widely used approaches, i.e. User-Based and Item-Based Collaborative Filtering (CF) as the baseline method to illustrate the effectiveness of our proposed method in recommending existing items. We also notice that in the field of RS, many state-of-the-art approaches have been proposed in recent years. It is fair to compare our work with these methods, but the complexity and unclear description in the original publications of these methods make it difficult to re-implement all of them. A better way to

do the comparison is to conduct the experiment with our method under the same settings used by the other methods, and compare our results against the reported results using these other methods. Although most methods report results in only one dimension of evaluation, it is reasonable to make such comparisons since we believe that in the evaluation dimension reported, these methods have the best results.

Impact of Sparsity. To investigate the impact of data sparsity, we first compare the prediction accuracy of our method with the two most widely used methods in practice, i.e. User-Based CF (UCF) (Resnick et al. 1994) and Item-Based CF (ICF) (Sarwar et al. 2001), using training data at different levels of sparsity. We randomly select a certain percentage of ratings as the testing set, and the remaining ratings serve as the training set. We introduce a variable *tp* to indicate what percentage of rating data is used as the test set. For example, *tp*=10% indicates 10% of the data is used as the test set, and the remaining 90% of the data is used as the training set. A higher value of *tp* indicates a higher sparsity level of the training set. We refer to our method of using critic review topics as the **CRT** method.

Figure 4.2 shows the MAE of the three methods using different *tp* values. The results show that as the percentage of data in the testing set increases, the prediction errors of all the three methods increase, but our CRT method always has a lower prediction error than the other two methods. Specifically, using 10%

of the data as the testing set, our CRT method reduces the errors of UCF and ICF by 7.93% and 3.27% respectively, while using 90% of data as testing set, the reduction in error becomes 9.02% and 9.31% respectively. That is to say, our CRT method is less affected by the data sparsity, and its strength in prediction accuracy is more salient in rating-sparse settings. Our method has lower prediction errors even in sparse settings, since it uses the additional topic information from critic reviews to eliminate the dependency on user ratings and it fully utilizes sparse ratings.

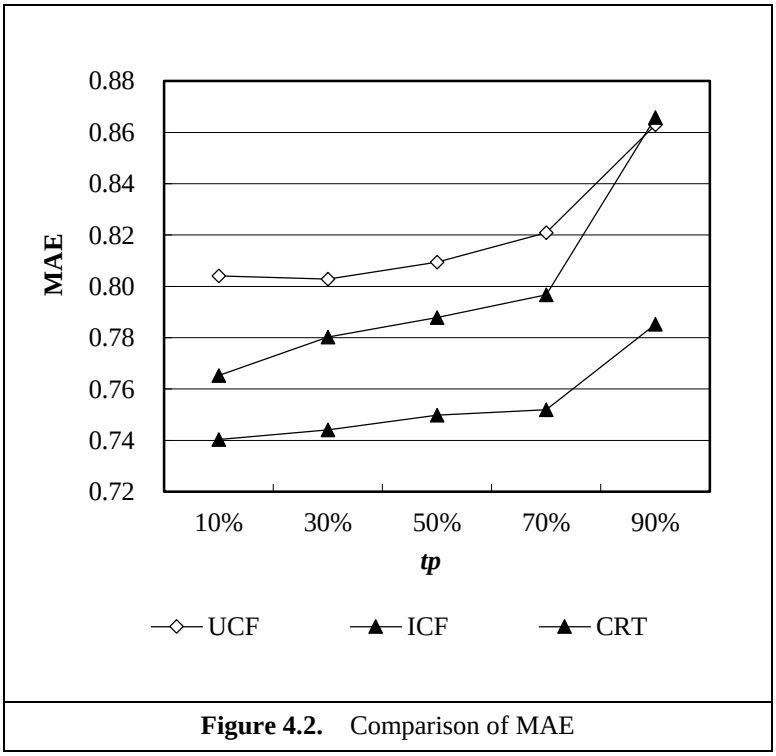


Figure 4.2. Comparison of MAE

Comparison of Efficiency. Our proposed method is efficient since it reduces the dimension of the original matrix, and the main process of every rating prediction is to compute the dot product of two vectors in *n* dimensions (in our

experiment n=68). To show the efficiency of our CRT method, we compare the time cost of the three methods in the above experiment. Since the efficiency of the online component is much more important than the offline procedures, we only consider the time cost of the online component (i.e. the Recommender) of our method in predicting the ratings in the testing set. For UCF and ICF, we also assume that the user similarities and item similarities can be computed offline and only account for the time cost by predicting the ratings in the testing set.

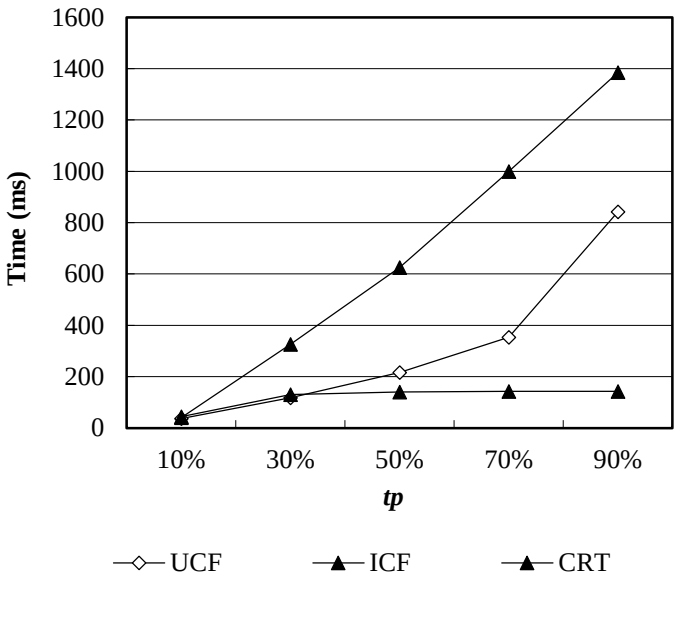


Figure 4.3. Comparison of Efficiency

The results are shown in Figure 4.3. From the results, we can see that as the size of testing set increases, the time cost of UCF and ICF increases rapidly, while our method still remains efficient for large testing sets. When run on testing set with 90% of data, our method costs only 10.32% of the time used by ICF, and 16.96% of that of UCF.

Comparison of Prediction Errors. In this experiment, we compare the prediction errors of our CRT method with other state-of-the-art methods that have been reported to have good results in rating prediction. These methods for comparison are:

- CBS (Xue et al. 2005): This is a cluster-based smoothing method. It fills in the missing values by using other users' ratings in the same user cluster.
- WLR (Srebro and Jaakkola 2003): This method uses weighted low-rank approximation to fill in the missing values.
- CBT (Li et al. 2009): This method expands the codebook to reconstruct the rating matrix that is used to fill in the missing values.
- SVD++ (Koren 2008): This method reduces the dimension of the original matrix through Singular Value Decomposition. It also integrates user feedback.
- TyCo (Cai et al. 2014): This method applies LDA to model keywords of movies and then constructs item typicality for further recommendation.

To make our result comparable, similar as for (Cai et al. 2014) and (Li et al. 2009), we randomly select 500 users from the dataset, then use the first 100, 200 and 300 of them to form the training sets, named ML100U, ML200U and

ML300U respectively. The last 200 users are used as testing set. For every user in the testing set, we keep 5, 10 and 15 ratings given by him in the training set, named as G5, G10 and G15 respectively. The training sets that have fewer users and fewer ratings from the test users are sparser. E.g. ML100U-G5 has the highest sparsity and ML300U-G15 has the lowest sparsity.

Table 4.4. Comparison of Prediction Errors (MAE)

	ML100U			**ML200U**			**ML300U**		
	G5	**G10**	**G15**	**G5**	**G10**	**G15**	**G5**	**G10**	**G15**
CBS	0.874	0.845	0.839	0.871	0.833	0.828	0.870	0.834	0.819
WLR	0.915	0.875	0.89	0.941	0.903	0.883	1.018	0.962	0.938
CBT	0.840	0.802	0.786	0.839	0.800	0.784	0.840	0.801	0.785
SVD++	0.925	0.911	0.916	0.881	0.815	0.812	0.885	0.815	0.802
TyCo	0.830	0.799	0.777	0.830	0.775	0.775	0.814	0.762	0.760
CRT	**0.788**	**0.783**	**0.774**	**0.782**	0.775	**0.768**	**0.774**	0.767	0.760

Table 4.4 shows the comparison of our CRT method with the state-of-the-art methods on MAE. The results of other methods are reported in (Cai et al. 2014) and (Li et al. 2009). The results show that in most settings, our method has lower prediction errors than other methods. Excluding our method, the TyCo method has the best results among the others. In rating-dense settings, e.g. ML300U-G10 and ML300U-G15, the prediction errors of our CRT method are very close to those of TyCo, while in rating-sparse settings, e.g. all ML100U and all G5, our method outperforms TyCo. The results are consistent with our findings in the previous experiment that our method has strength in sparse settings. Specifically, our CRT method reduces the prediction errors of TyCo

using ML100U-G5, ML200U-G5 and ML300U-G5 by 5.06%, 5.78% and 4.91% respectively.

Comparison of Ranking Accuracy. In this experiment, we test the ranking accuracy of our CRT method, and compare the results with those reported by state-of-the-art methods that have proven to have good performance in ranking the recommended items. The methods to be compared with are:

- ASSOC (Deshpande and Karypis 2004): This method uses the association among items to perform the top N recommendations.
- FREQ (Sueiras et al. 2007): This method builds a model based on the hitting-frequency to predict the user preference.
- PMF (Salakhutdinov and Mnih 2008): This method employs Probabilistic Matrix Factorization to utilize the relationship among users, items and ratings.

Table 4.5. Comparison of Ranking Accuracy (NDCG@*k*)

	G5			**G10**			**G15**		
	NDCG @1	**NDCG @3**	**NDCG @5**	**NDCG @1**	**NDCG @3**	**NDCG @5**	**NDCG @1**	**NDCG @3**	**NDCG @5**
ASSOC	0.529	0.542	0.560	0.597	0.593	0.595	0.615	0.610	0.627
FREQ	0.642	0.600	0.596	0.636	0.607	0.610	0.638	0.618	0.632
PMF	0.635	0.612	0.623	0.644	0.646	0.654	0.696	0.689	0.698
CRT	**0.710**	**0.691**	**0.681**	**0.709**	**0.694**	**0.679**	**0.712**	**0.692**	0.673

Following the experiment in (Xin et al. 2011), we randomly choose 600

users to form the training set and the remaining 343 users are put in the testing set. For every user in the testing set, 5, 10 and 15 ratings from him are given in the training set, named as G5, G10 and G15 respectively. The fewer the ratings given to the training set, the sparser it is.

The results of NDCG@1，NDCG@3 and NDCG@5 using different training and testing sets are shown in Table 4.5. The results of other methods are reported in (Xin et al. 2011). The results show that in all settings except G15-NDCG@5, our CRT method has better results, especially in the sparse settings like G5. The results are in line with our previous findings that the advantage of our CRT method is more salient in sparse settings. Our CRT method increases the NDCG@1, NDCG@3 and NDCG@5 of PMF using the sparsest training set (i.e. G5) by 11.81%, 12.91% and 9.31% respectively.

Comparison of New Item Recommendation. One of the key features of our method is that it supports new item recommendation, while all the above-mentioned state-of-the-art methods cannot work under cold start settings with new items. To illustrate the effectiveness of our CRT method in recommending new movies, we compare with another method, TSCF (Spaeth and Desmarais 2013), that computes the text similarity between the item profiles (here we use the movie plot summaries in IMDB[5]), and then performs CF recommendation.

[5] http://www.imdb.com

In order to simulate the new movies, we randomly select 200, 400 and 600 movies as new movies for testing, named as ML200M, ML400M and ML600M respectively, and the remaining movies are used as existing movies for training. For movies in the testing set, none of their ratings are given in the training set. Since the scale of the predicted ratings for new movies given by our method is different from that of the users' real ratings, we do not compare the prediction errors here and only report the results of NDCG@*k* that are shown in Table 4.6. The results indicate that as the proportion of new movies increases, the ranking accuracy of both methods decreases, but our CRT method always performs better than the TSCF method. The results prove that our CRT method is effective in new item recommendation.

Table 4.6. Comparison of New Item Recommendation (NDCG@*k*)

	ML200M			**ML400M**			**ML600M**		
	NDCG @1	**NDCG @3**	**NDCG @5**	**NDCG @1**	**NDCG @3**	**NDCG @5**	**NDCG @1**	**NDCG @3**	**NDCG @5**
TSCF	0.477	0.470	0.467	0.465	0.468	0.460	0.458	0.459	0.457
CRT	**0.501**	**0.510**	**0.505**	**0.487**	**0.494**	**0.495**	**0.480**	**0.485**	**0.489**

4.6. Conclusion

In this study, we propose a novel content-based recommendation framework. A distinct feature of our method is that it incorporates the topics inferred from the external critic reviews of items to boost the cold start recommendation. We employ an advanced semi-supervised topic modeling approach, i.e. PLDA,

which is able to uncover the global shared latent topics as well as the topics under each well-structured item attribute, to learn and infer the topic distribution of the critic reviews. We also adapt NMF to our context by redefining the error function to fully utilize the user ratings and topic distribution of critic reviews. The topics inferred from is critic reviews are better representations of the items since it covers more characteristics of the items and reflect more aspects of user tastes. By fully utilizing the user ratings and the inferred topics, our method alleviates the dependency on user ratings and enables high quality recommendations even under cold start settings with new items. The adaption of NMF lowers the dimension of the original rating matrix, which contributes to high efficiency. The results of the experiment show that our proposed method is scalable and outperforms the current state-of-the-art methods in terms of prediction accuracy and ranking accuracy, and the advantage of our method is more salient in rating-sparse settings. Our method also generates high quality new item recommendations which is not supported by many current state-of-the-art methods.

There are some limitations of our work. First, some off-line processing procedures (e.g. topic learning) are time consuming. In future work, a parallel computation framework (e.g. MapReduce) can be adopted to accelerate the computation for large scale applications. Second, we only use one kind of attribute (i.e. the genre of movie) to supervise the topic learning and inferring.

Future work could use more attributes and explore how to integrate topics under different attributes.

Although we have focused on the domain of movies in this study, our method is generally applicable to any other domain of consumer products where critic reviews are available. One possible extension to our work is to see whether our method can be applied in cross-domain recommendation.

CHAPTER 5. STUDY ON FUNCTIONALITY-BASED MOBILE APP RECOMMENDATION BY IDENTIFYING FUNCTIONAL ASPECTS FROM USER REVIEWS

5.1. Introduction

Accelerated by the popularity of smart phones, the mobile application (or app for short) market is growing explosively. For instance, the Apple App Store provides more than one million apps in 24 categories for users in 155 countries around the world[6]. On one hand, tens of thousands of new apps are continuously being released in app stores, but most of them can hardly be reached by users via keyword searches; on the other hand, it has been a significant challenge for users to find the apps they need in such crowded app stores. Therefore, it is necessary to have effective mechanisms to help users discover relevant apps among the overwhelming number of alternatives.

To alleviate the new item discovery problem, many industry solutions, such as the personalized recommender systems (RS), for other consumer product domains, e.g. books, movies, music etc., have been proposed. These solutions mostly deal with the new item problem by recommending items that are similar to those the user has selected (Celma et al. 2005; Rafailidis et al.

[6] http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html

2014; Schwab et al. 2001; Semeraro et al. 2009). While the general goal of mobile app recommendation is similar to those in traditional domains – to guide users to items that are relevant to their interests, there are unique features of mobile apps that make the solutions in traditional domains less effective in the app domain.

One of the most important characteristics of mobile app selection is that it is based more on the apps' functionalities than the users' taste. For instance, a user who likes the movie *Titanic* may be glad to watch another romantic movie similar to *Titanic*; however, if a user has installed an app providing particular functionality, e.g. whether forecast, he/she needs no more similar apps with the same functionality of whether forecast, unless they provide additional functionalities. If existing recommendation techniques are directly applied in the app domain, users may be end up receiving a mass of redundant app recommendations providing similar functionalities.

Moreover, the most widely used recommendation techniques, i.e. Collaborative Filtering (CF) (Sarwar et al. 2001) and Content-based Filtering (CB) (Pazzani and Billsus 2007), usually generate recommendations based on user ratings. In the app domain however, rating values indicate more about users' evaluation of the non-functional aspects (ease of use, UI design, power consumption etc.) of the app, but can hardly reflect the users' functional requirements. For example, even if a user gives a very low rating to an app

providing weather forecast, we can only say the user is not satisfied with this app (maybe because it is power consuming), but we cannot deny the fact that this user needs the functionality of weather forecast, since he has been attracted by the described functionalities of this app and has decided to install it. Therefore, when applied in the app domain, traditional techniques fail to reveal the detailed functionalities inside apps, and lack the ability to capture users' functional requirements, which may worsen the quality of recommendations.

Recently, an increasing amount of research has paid attention to mobile app recommendation. These works have enjoyed varying degrees of success by either adapting traditional recommendation techniques to the app domain (Bhandari et al. 2013; Lin et al. 2013; Yan and Chen 2011) , or considering additional dimensions of apps (e.g. context information) (Böhmer et al. 2010; Karatzoglou et al. 2012; Shi et al. 2012). However, the redundancy problem in app recommendation has received scant attention from researchers, and there has been no reported work on app recommendations that considers user requirements at the functionality level.

To bridge this gap, in this study, we propose a functionality-based recommendation solution that is able to provide more accurate and more diverse app recommendations by drilling down into users' functional requirements. In our proposed solution, a mobile app is modelled as a collection of different functionalities, and user requirements are modelled at the functionality level.

We first predict what new functionalities a given user most likely needs based on other users' usage patterns, and select a collection of apps containing these new functionalities as recommendations. If there are similar apps providing overlapping functionalities in this collection, we only recommend the top app that has the best quality, therefore truly capturing users' functional requirements and avoiding redundant recommendations.

We achieve our goal by solving three important problems. First, given an app, we need to know what functionalities it has. Although some functionalities are explicitly stated in the apps' descriptions, they are embedded in short text blocks and are hard to be identified from the descriptions alone. We note however, that the functionalities of an app may be repeatedly mentioned in the app's user reviews. In addition, user reviews may also contain other implicit functional aspects that are not stated in the descriptions but are useful for modeling user requirements. Therefore, one main feature of our solution is to obtain functionalities of apps by mining textual user reviews. To accurately extract both explicit and implicit functional aspects of apps from noisy review content, we propose a simple but effective approach by combining app descriptions and user reviews.

Second, user requirements should be properly modelled. We propose a graph-based approach called *AppRank* to utilize the propagation of user requirements at the functional level, and employ a two-stage random walk

process to predict new functionalities for the users.

Third, we need to rank and select good apps from similar candidates providing overlapping functionalities to avoid redundancy. Our AppRank method introduces a competition mechanism to distribute weights among similar apps, which gives priority to apps of higher quality.

To the best of our knowledge, this is the first work to consider users' functional requirements in mobile app recommendation. We prove the possibility of extracting app functionalities from textual user reviews, and we also propose an effective solution that enables functionality-based app recommendation. The results of experiments conducted on a real-world mobile app dataset show that our proposed method outperforms baseline methods in terms of stability against data sparsity, ranking accuracy in top N recommendations, overall ranking correctness and recommendation diversity.

The remainder of this chapter is organized as follows: first we review related works in literature. Next we describe the intuition behind our proposed solution and first provide an overview, followed by a more detailed elaboration. Then we evaluate our solution and present the results of our evaluation. Finally, we discuss the contribution of our work to the field and possibilities for further work.

5.2. Related Work

Recently, researchers have started paying attention to mobile app recommendation, and an increasing amount of research on app recommendation is being done. In the following, we will review related work on mobile app recommendation, and discuss related work on page-rank based methods which will be adapted in our method to discover new functionalities for users.

5.2.1. Mobile App Recommendation

A few studies propose to extend traditional recommendation algorithms and to adapt them into the app domain. For example, *AppJoy* (Yan and Chen 2011) replaces the user ratings in traditional RS with usage scores composed by recency, frequency and duration, and then performs item-based CF recommendation. Bhandari et al. (2013) adapt graph-based recommendation for app discovery, aiming at improving novelty. Lin et al. (2013) propose to extend model-based RS by constructing latent user models from apps' twitter followers, addressing the cold-start problem of app recommendation. Hybrid methods are also existing. For example, Xia et al. (2014) report a multi-object approach to evolve existing mobile app RSs. Although these solutions have proven to be effective to some extent in recommending apps, they do not consider much about the unique characteristics of apps.

Noticing this limitation, some researchers have shifted their focus to a

unique characteristic of mobile apps – context, and a few context-aware systems have been proposed in the app domain. Such systems record users' context information, e.g. physical location, at a particular time and then enhance app recommendation by exploiting the collected context information (Liu et al. 2013). For example, Böhmer et al. (2010) explored the design space for context-aware app recommendation, and developed a prototype app RS on Android platform called *Appazaar*. The *Djinn* model introduced by Karatzoglou et al. (2012) utilizes the user-app-context relationship using tensor factorization, providing a new context-aware CF approach for app recommendation. Shi et al. (2012) also apply tensor factorization to integrate implicit feedback data with contextual information, and they propose to generate app recommendations by optimizing the ranking (i.e. MAP). Context-aware app RSs are highlighted since they take into account one important feature of mobile app, i.e. context information. Such systems show better performance than traditional methods in recommending apps. However, context information is very difficult to collect, due to privacy concerns and other constraints. It has been a significant limitation of context-aware systems.

To conclude, existing works on mobile app recommendation do consider some unique features in the app domain; however, no reported work has been found to recommend apps at the functionality level and to avoid redundant recommendations. These gaps will be addressed with our proposed method.

5.2.2. PageRank-Based Methods

PageRank (Page et al. 1999) is a graph-based ranking algorithm proposed by Google, and has been successfully applied in analyzing the link-structure of the World Wide Web. The objective of PageRank is to determine the importance of a given webpage on the web hyperlink structure. The basic assumption of PageRank is that a web page is more likely to be authoritative if it is linked to by many other authoritative pages. The implementation of PageRank is based on a "voting" mechanism. If a webpage links to another page, it denotes a vote to that target page. Moreover, the weight of the vote is determined by the importance of the webpage which gives the vote. Finally, the greater the weight of the vote a webpage receives, the more important it is. The final weight, i.e. the PageRank score, of a webpage is determined by a random walk process which iterates the voting process throughout each node in the graph until it converges.

Based on PageRank, many variants in different domains have been proposed. For example, Mihalcea and Tarau (2004) propose a graph-based ranking model called TextRank for keyword and sentence extraction in the domain of natural language processing. In the TextRank model, each word is modelled as a vertex, and the edges in graph represent the concurrence of words in the document. Jeh and Widom (2003) introduce the personalized PageRank vector into the original model and propose a personalized version of PageRank,

which is able to capture user preference. FolkRank, proposed by Hotho et al. (2006), is an adaption of the PageRank algorithm for folksonomy ranking and searching. FolkRank employs a differential approach to compute FolkRank score by taking the difference between the personalized PageRank score and the original PageRank score.

Our proposed method combines and adapts TextRank and FolkRank in the context of mobile app functionality prediction, and we call it *AppRank*. The details of our adaption will be provided in the ensuing sections.

5.3. Intuition and Overview

We are interested in helping mobile app users discover new functionalities they may need, and recommending apps that can truly meet their requirements. Our proposed method is motivated by users' real-life behavior of selecting mobile apps. When choosing an app to install, a user usually first considers whether the app provides the functionalities he/she needs by reading the app's description. If there are many alternatives providing similar functionalities, the user may try each of them and evaluate them on other non-functional aspects (e.g. UI design, ease of use, power consumption), and then select the most preferred one to use. At a high level, our method automates this process through three main steps: (1) knowing all the functionalities provided by the apps that a user has been using; (2) predicting what other functionalities this user may need; and (3) helping the

user select better apps providing these desired functionalities.

For example, let's assume that the target user has installed an app providing weather forecast and airline information in his/her mobile phone. By analyzing other users' usage patterns, we find that users who use apps providing weather forecast or airline information may also use apps providing navigation that the target user has not installed. We then select a set of apps providing navigation as recommendation candidates. To avoid generating redundant recommendations, we rank the candidate apps providing similar functionalities and only select the top one that has the best quality as recommendation.

One of the most outstanding features that differentiate our method from existing works is that we generate recommendations at the functionality level, truly capturing users' functional requirements. To achieve our goal, the most important problem we need to solve is obtaining the functionalities of each app. An intuitive solution is to extract app functionalities from their textual descriptions. But we quickly realize that descriptions are short texts wherein functionalities may not be repeatedly stated. Most of the traditional keyword extraction techniques (usually based on term frequency) are designed for long articles, which may not be effective when applied to app descriptions. Fortunately, researchers have found that item features are frequently mentioned in customer reviews (Hu and Liu 2004). This motivates us to obtain app functionalities from user reviews. However, it is common to have user reviews

containing a lot of noisy content that is not relevant to the app functionalities. In order to filter out noisy content, we propose to use the apps' description content as a reference to construct a vocabulary, and perform frequency analysis on the user reviews, which helps to extracting high quality feature words and phrases related to the app functionalities. Next, after acquiring the app functionalities, we propose a graph-based ranking method to discover new functionalities for the users by propagating their requirements in a functionality co-occurrence graph. We also intelligently filter out apps with overlapping functionalities, therefore capturing user requirements and addressing the redundancy problem. The details of our proposed solution will be introduced in the following section.

5.4. Solution Details

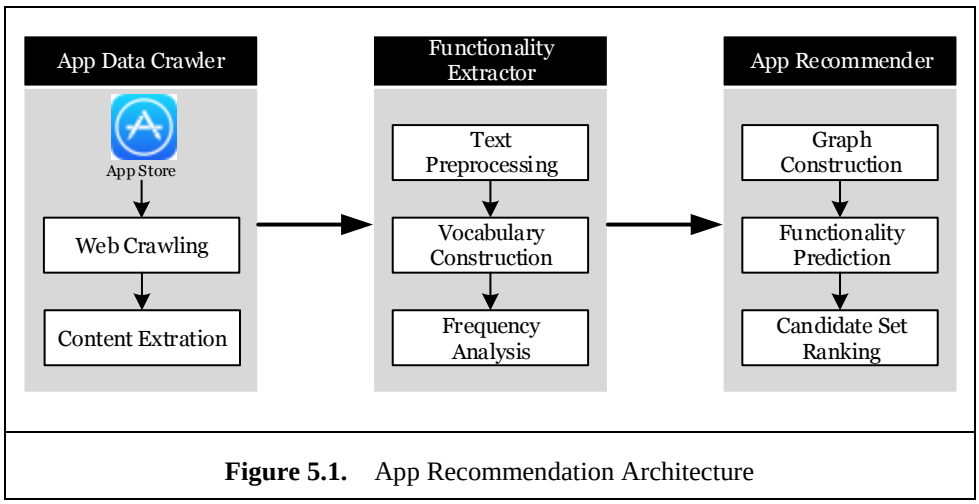


Figure 5.1. App Recommendation Architecture

In this section, we will first show the architecture of our proposed solution, followed by the details of each component in the architecture.

Our proposed app recommendation architecture is shown in Figure 5.1. There are three main components in the architecture: *App Data Crawler*, *Functionality Extractor* and *App Recommender*. We use the App Data Crawler to collect app descriptions and corresponding user reviews. From the collected data, app functionalities are then extracted by the Functionality Extractor. Finally, the App Recommender predicts new functionalities for the user, selects candidate apps to recommend, and intelligently filters out apps with overlapping functionalities. More details of each component will be given in the ensuing sections.

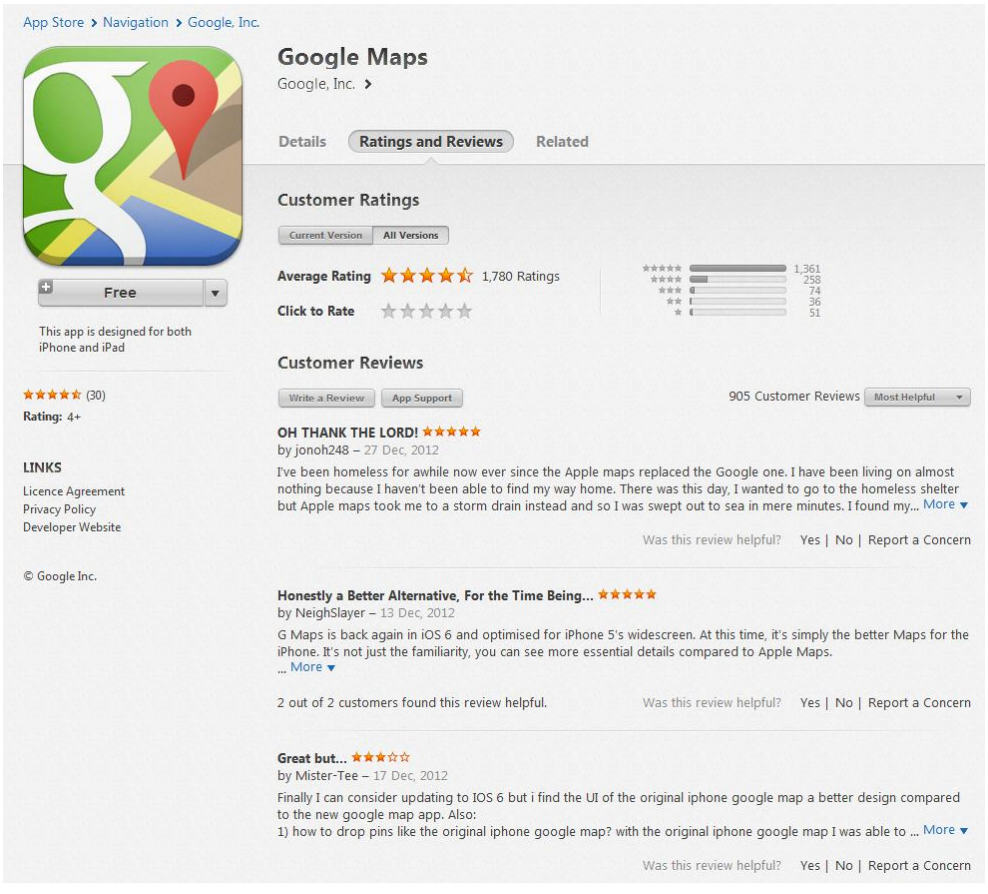


Figure 5.2. User Reviews in Apple App Store

5.4.1. App Data Crawler

The main task of the crawler is to collect web pages containing app descriptions and user reviews from the app store. Figure 5.2 shows one of the app web pages. Since the needed content is embedded in HTML files, we develop an extractor to extract the textual content of app descriptions and user reviews. User ratings associated with reviews are also isolated.

5.4.2. Functionality Extractor

Text Preprocessing. The inputs of the Functionality Extractor are the textual content of each app's descriptions and user reviews. We use the Stanford Core Natural Language Processing toolkit[7] to perform text preprocessing, including tokenization (breaking up text into words), Part-of-Speech (POS) tagging (e.g., noun, verb, adjective), lemmatization (converting words to their based forms, e.g. "emails" and "emailing" are converted to "email"), and removing stop words (i.e. non-content words that appear too frequently in all apps, like "a", "the").

Vocabulary Construction. In order to get rid of noisy content that is irrelevant to app functionalities in user reviews, we need to control the size of the vocabulary. Although app descriptions may be too short for functionality extraction, the vocabulary used in app description is more formal and more

[7] http://nlp.stanford.edu/software/corenlp.shtml

relevant to app functionality. It turns out that the app description can be a good source for constructing a vocabulary. After looking at the data, we notice that, most app functionalities are in the form of single nouns (e.g. navigation), noun phrases (e.g. flight information) and verb-object phrases (e.g. read book). We then aggregate all app descriptions from which we only keep the single nouns, two-gram nouns and two-gram verb-object phrases in the vocabulary. We refer to a single word or a 2-gram phrase in the vocabulary as a *functional aspect*. We also remove those aspects that are too rare, i.e. appearing less than 10 times, from the vocabulary. We believe the constructed vocabulary is able to cover most functional aspects of apps.

Frequency Analysis. In this step, we perform frequency analysis on the app descriptions and user reviews, and extract the most frequently mentioned functionalities for each app. We denote the vocabulary as *V*. For each aspect $w \in V$, we calculate its weight that indicates its representativeness of app *a* as:

$$Weight_{w,a} = (m \times dsf_{w,a} + rvf_{w,a}) \times \log \frac{N}{af_w}.$$

where $dsf_{w,a}$ is the frequency of aspect *w* in app *a*'s description; and $rvf_{w,a}$ is the number of app *a*'s user reviews that mention aspect *w*. $\log \frac{N}{af_w}$ is the inverse app frequency that indicates the aspect's discriminating power, where *N* is the total number of apps, and af_w is the number of apps that contain aspect *w*.

The proposed aspect weighting scheme uses a linear combination of the description frequency *dsf* and the review frequency *rvf*, and multiplies *dsf* by *m* to emphasize those aspects appearing in the description. Actually we can regard the app description as an important piece of review. If an aspect is mentioned one time in the description, it is as important as being mentioned by *m* users. We use the number of reviews that contain the aspect instead of using the frequency of the aspect in all reviews, because we believe an aspect mentioned by 10 users is more important than an aspect mentioned 10 times by one user. The proposed weighting scheme is able to consider the situation where the user reviews are not sufficient. When the number of reviews is less than *m*, *dsf* dominates the aspect weight, therefore avoiding bias caused by a small number of reviews. Similarly, if an app's description is extremely short and does not contain informative content, *rvf* allows us to find out frequently mentioned functional aspects that are not explicitly stated in the description (i.e. *dsf* is zero).

After frequency analysis, we are able to obtain the functional aspects for each app by selecting the top 50 aspects having the highest weights.

5.4.3. App Recommender

Graph Construction. One of the main tasks of the recommender is to predict new functionalities for the target user. We employ a graph-based ranking approach which is able to propagate users' functional requirements in the

functionality graph. The first step is to construct the functionality graph that captures the co-occurrence of functionalities based on global usage patterns from all users.

Let $G=(V, E)$ be a directed graph with a set of vertex V and a set of edges E. A vertex V_w denotes a functionality w, and an edge $E_{i,j}$ from vertex i to j denote an association from functionality i to functionality j, which means if i appears, j usually appears as well. We use a directed graph instead of an undirected one because association between two functionalities is asymmetric. For example, users who need navigation may also need weather forecast, but users who need weather forecast may not need navigation.

We use the well-known constraints in association rule mining, i.e. support and confidence, to determine whether to add an edge into the graph or not. Support is a measure of usefulness of the association. An association having too low support may happen just by chance. In our context, support of an association $i \Rightarrow j$ is defined as:

$$Support(i \Rightarrow j) = \frac{|U(i, j)|}{|U|}.$$

where $|U(i,j)|$ is the number of users who install apps with functionality i and apps with functionality j, $|U|$ is the total number of users. We are interested in the association of functionalities in different apps but not in the same app. If a user installs only one app with both functionality i and j, he will not be included

in $U(i,j)$. A support value of 0.4 means that 40% of the users have both functionality *i* and *j* in their mobile devices.

Confidence is a measure of certainty of the association. Confidence of the association $i \Rightarrow j$ can be regarded as the conditional probability of $P(j \mid i)$. In our context, it is defined as:

$$Confidence(i \Rightarrow j) = \frac{|U(i, j)|}{|U(i)|}.$$

where $|U(i,j)|$ is the number of users who install apps with functionality *i* and apps with functionality *j*, $|U_i|$ is the number of users who install apps with functionality *i*. A confidence value of 0.4 means that among the users who have functionality *i* in their mobile devices, 40% of them also have functionality *j* in their mobile devices.

An edge $E_{i,j}$ is added into the graph if the association $i \Rightarrow j$ satisfies both a minimum support threshold and a minimum confidence threshold, which is 0.1 and 0.4 respectively in our implementation.

Functionality Prediction. With the constructed functionality graph, we are able to make predictions of new functionalities for a given user. Similar to Jeh and Widom (2003), we follow a two-stage random walk process to propagate user requirements to new functionalities. At the first stage, we run the original PageRank random walk model on the functionality graph. Let $In(V_j)$ be the set

of vertexes pointing to V_j, and Out(V_i) denote the set of vertexes pointed by V_i. The score of each vertex *j* at the first stage is given by:

$$PR(V_j) = (1-d) \times p(V_j) + d \times \sum_{V_i \in In(V_j)} \frac{PR(V_i)}{|Out(V_i)|}.$$

where a user follows the association to install a functionality with probability *d*, and jumps to a completely new functionality with probability 1-*d*. In our implementation, we use the same value of *d* as the original model, which is 0.85. $p(V_j)$ indicates the user's preference for functionality V_j. At the first stage, we run the non-personalized PageRank, so $p(V_j)$ is set to 1 for every vertex. We iterate the computation of *PR* score for each vertex until it converges.

The *PR* scores given at the first stage indicate how often each functionality co-occurs with other functionalities. However, what we want to know is how the user requirements may flow to other vertex along the edges of the graph. Therefore, at the second stage, we run the personalized PageRank, in which $p(V_j)$ is given a large value (we set it as $|V|$) if the functionality V_j has been used by a user. Similarly, we iterate the computation of the personalized score *PR*' (V_j) for each vertex V_j until it converges. Then we employ a differential approach to obtain ΔPR:

$$\Delta PR(V_j) = PR' - PR.$$

ΔPR indicates the weights propagated from the functionalities that have

been used by the user. It can be regarded as a measure of how likely the user needs the new functionalities. In next section, we will introduce how to utilize ΔPR in app recommendation.

Candidate Set Ranking. With ΔPR, we are able to predict new functionalities for a given user, and then retrieve candidate apps that contain these new functionalities. However, the candidate set generated in this way may contain many apps with overlapping functionalities. To avoid redundant recommendations, we need to rank apps from the candidate set, with two objectives: (a) to promote apps with better quality; (b) to promote apps providing more functionalities needed by the user.

To achieve these objectives, we come up with a competition mechanism to distribute ΔPR of all functionalities to the apps that provide these functionalities. First, for each functionality, we search for all apps that provide this functionality. Second, we rank these app based on the number of users who have installed them, and only the one that has the highest ranking can be awarded the ΔPR of the functionality. Here our assumption is, if two apps provide similar functionalities, the one installed by more users usually has better quality. Finally, for each app, we aggregate the ΔPR it wins from all functionalities it provides, to obtain the AppRank Score, that is:

$$AppRank(App_a) = \sum_{V_i \in Win_a} \Delta PR(V_i).$$

where Win_a is the set of functionalities for which App_a ranks higher than other apps.

We select the top K apps that have the highest AppRank scores as recommendations. Our completion mechanism allows only one app to obtain the ΔPR for each new functionality, therefore avoiding redundant recommendations. The AppRank score uses the summation of ΔPR from different new functionalities, which gives priority to the apps that provide more needed functionalities.

5.5. Experiment and Results

In this section, we describe the experiment we conducted to evaluate the effectiveness of our proposed solution. First, we introduce the evaluation metrics used in the experiment. Then we describe the experiment setup. Finally, we will report the results, including functionality extraction, impact of sparsity, ranking accuracy, and recommendation diversity.

5.5.1. Evaluation Metrics

To evaluate our proposed method, we compare our method with other state-of-the-art recommendation techniques on several evaluation metrics. Specifically, we will evaluate the ranking accuracy and recommendation diversity. For ranking accuracy, we use two metrics. The first one is Recall@*k*, which is defined as:

$$Recall@k = \frac{\#\text{liked items in top } k \text{ recomendations}}{\#\text{liked items}}.$$

For ranking accuracy, recall is usually measured with another metric – precision, which indicates what proportion of recommended items are liked by the users. However, since most items are unrated, it is hard to say whether the users dislike the unrated items, or they just do not know these items. Therefore, we only use the recall which we think is more pertinent, since it only considers the liked items.

In addition to Recall@*k* that measures the ranking accuracy for the top *N* recommendations, we use another measure — NDCG (Herlocker et al. 2004) to evaluate the overall ranking accuracy. NDCG is defined as:

$$NDCG = \frac{1}{|U|}\sum_{u \in U} Z_u \sum_{p=1}^{m} \frac{2^{r_{u,p}} - 1}{\log(1+p)}.$$

where U is the set of users; Z_u is a normalization factor to guarantee that for perfect ranking the NDCG value is 1; p is the position of the recommended item in the list; m is the size of candidate items; and $r_{u,p}$ is the rating given by the user u to the item at position p.

Recommendation diversity is measured as 1 minus Intra-List Similarity (Järvelin and Kekäläinen 2002) that is defined as:

$$ILS = \frac{\sum_{Rec_i \in Rec} \sum_{r_a \in Rec_i} \sum_{r_b \in Rec_i, r_b \neq r_a} Sim(r_a, r_b)}{2|Rec|}.$$

where *Rec* is the set of recommended items to all users; Rec_i is the list recommended items for user *i*; r_a and r_b are two different items in user *i*'s recommendation list; and $Sim(r_a, r_b)$ measures the content similarity between item r_a and r_b, which is the proportion of overlapping functional aspects of two apps in our implementation.

5.5.2. Experiment Setup

The data we use in the experiment is crawled from the Apple App Store (U.S.)[8]. We construct the vocabulary based on the textual descriptions of 10530 popular apps evenly distributed in 22 categories. The constructed vocabulary contains 20690 words and phrases. Our constructed dataset for evaluation contains 66543 ratings on a scale of 1-5 given by 1879 users to 2213 apps. The sparsity level (i.e. the percentage of empty entries in the user-app rating matrix) of the dataset is 98.39%. 1202 of the apps in the dataset are free, and the remaining 1101 apps are paid. The distribution of app categories in our dataset is shown in Figure 5.3. In the dataset, each user has rated at least 5 free apps and 5 paid apps. On average, each user has rated 20 free apps and 15 paid apps. For each app in the dataset, we collected a maximum of 500 user reviews. On average, each app had 442 reviews.

[8] https://itunes.apple.com/us/genre/mobile-software-applications/id36?mt=8

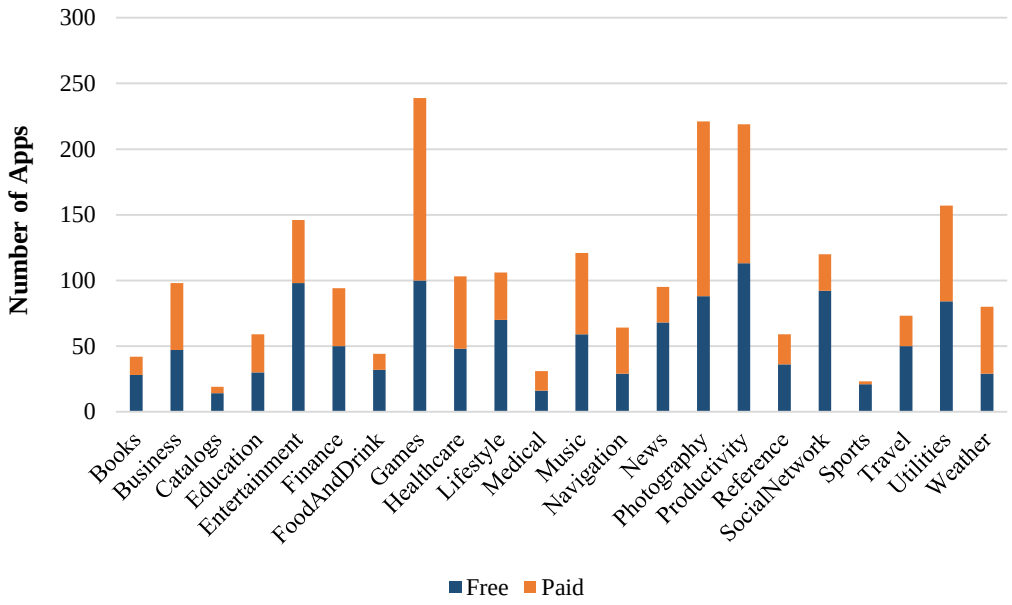


Figure 5.3. App Category Distribution

5.5.3. Experiment Results

Table 5.1. Extracted Functionalities

App Name	Functionalities
Dropbox	doc, file, space, photo, video, computer iphone, access file, share link, access photo, video device, share photo, attachment
WhatsApp	message, massager, chat, group, contact, friend, address book, chat history, friend world, send message, group chat, voice note
Kindle	book, newspaper, textbook, magazine, reader, bookmark, reading, reading experience, book mark, read book, pdf, dictionary
Gmail	mail, google, conversation, inbox, receive email, account support, attachment, get notification, account, mail app, contact, send email
YouTube	video, playlist, google, video playlist, list search, watch video, watch list, share video, channel, search video, share friend, entertainment

Qualitative Results for Functionality Extraction. To investigate the effectiveness of our method for extracting app functionalities, we select 5 popular apps and for each app, we only list the top 12 extracted functionalities using our method. The qualitative results are shown in Table 5.1.

From the results, we can see that most of the extracted functionalities are meaningful and reasonable. The quality of the extracted functionalities plays an important role in the whole solution, since the functionalities are the basis of further analysis for recommendation. The results show that our proposed method is effective in extracting app functionalities of good quality from user reviews, which guarantees the effectiveness of the whole solution.

Impact of Sparsity. In this experiment, we compare our method with other baseline methods for generating the top *N* recommendations using different training-test ratios. The baseline methods we compare with are: User-Based CF (UCF) (Resnick et al. 1994), Item-Based CF (ICF) (Sarwar et al. 2001), Content-based Filtering (CB), Non-negative Matrix Factorization (NMF) (Lee and Seung 1999), Regularized Singular Value Decomposition (RSVD) and its variant SVD++ (Paterek 2007). These methods are commonly selected for comparison in recommendation research.

We introduce a variable *tp* to indicate what percentage of the rating data is used as test set. For example, *tp*=10% indicates 10% of the data is used as test set, and the remaining 90% of the data is used as training set. A rating in the test set is converted into "like" if its value is larger than 3. We fix N=100, and vary the percentage of the test data *tp*=10%, 20%, …, 90%. The corresponding recall values are shown in Figure 5.4.

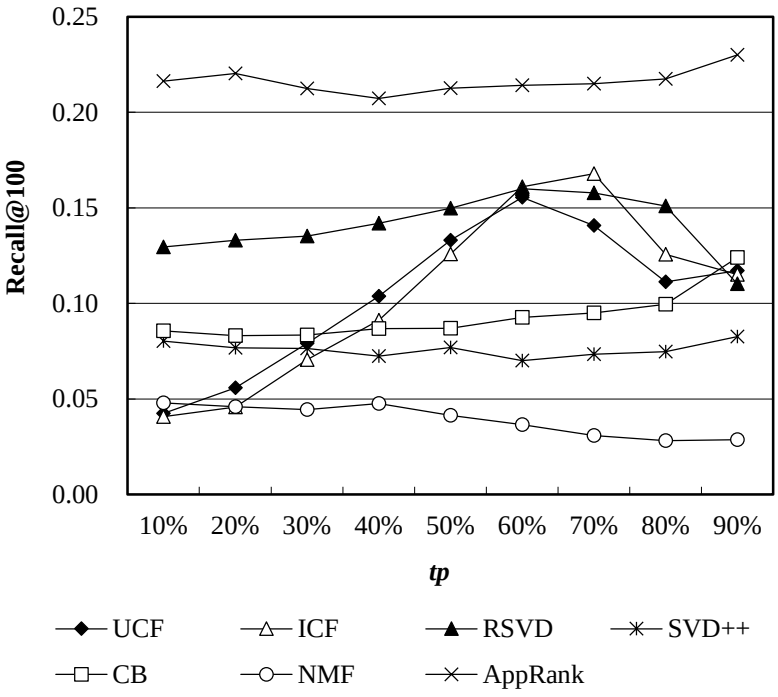


Figure 5.4. Comparison of Recall@100 with Different *tp*
(Higher recall indicates higher accuracy)

From the results, we find that the recall of all methods is generally low. One possible explanation for the low recall is that we tend to select active users when we construct the dataset, since we need a relatively dense dataset for evaluation, otherwise the results are very unstable. Some of these active users are app players, i.e. people who would like to try different kinds of apps for no particular reason, and therefore it is very difficult to predict their interests and requirements. In spite of the low recall, the results are still valid for showing the effectiveness of our proposed method when we look at the relative values.

The results show that our proposed AppRank method is less sensitive to training-test ratio compared to other methods, and it always outperforms other

methods on all *tp* values. As *tp* increases, less data is used for training, which means the sparsity level of the training set increases as well. Therefore, the results also show that our method is less sensitive to data sparsity, and its improvement is more salient in extremely sparse settings. Specifically, when *tp*=90%, our AppRank method increases the recall of the second best method, i.e. CB, from 0.12 to 0.23. The results prove the effectiveness of our AppRank method in alleviating data sparsity.

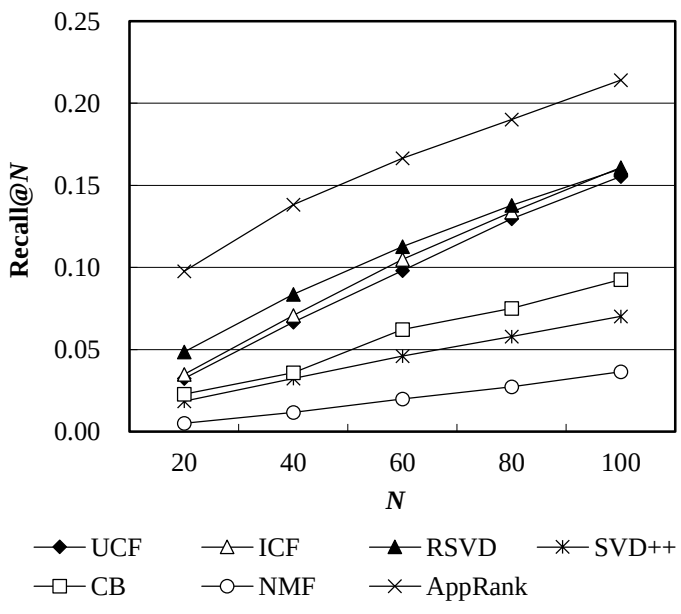


Figure 5.5. Comparison of Recall@N
(Higher recall indicates higher accuracy)

Comparison of Top N Recommendations. In this experiment, we fix the *tp* to 60%, where most methods have high recall, and vary the number of recommended apps *N*=10, 20, …, 100 to compare the recall of different methods for the top *N* recommendations. The results of the comparison in Figure

5.5 show that the recalls of all methods increase along with the *N*, and the recall of our method outperforms all other methods for different *N*. The results prove that our AppRank method has significant improvement on ranking accuracy for the top *N* recommendations.

Comparison of Overall Ranking. In this experiment, we still fix the *tp* to 60% and compare the NDCG values of different methods to investigate the correctness of overall rankings for all candidate items. From the results shown in Figure 5.6, we can see that our proposed AppRank method has the highest NDCG value, and it increases the NDCG value of the second best method, i.e. RSVD, by 14.27%. The results prove that our method is effective in improving the correctness of the overall ranking for all candidate apps.

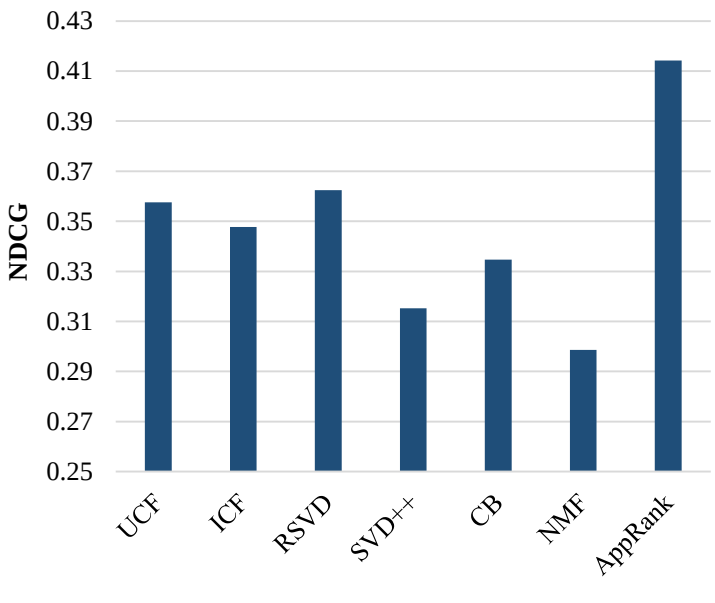


Figure 5.6. Comparison of NDCG
(Higher NDCG indicates higher accuracy)

Comparison of Recommendation Diversity. We find that at less sparse settings, generally all methods are able to generate diverse recommendations. However, when the training data becomes sparse, the diversity of some methods drops down. We set *tp*=90%, and compare the diversity of the top 5 and top 10 recommended apps of different methods. The results are shown in Figure 5.7. From the results, we can see that the diversity of the top 5 and top 10 recommended apps generated by our method remains high, which is 0.9913 and 0.9916 respectively. However, for UCF, ICF and CB, the diversity is significantly lower. For instance, the diversity of the top 5 and top 10 recommended apps generated by UCF is only 0.8715 and 0.9164 respectively. The results prove that our method is less sensitive to data sparsity in terms of recommendation diversity.

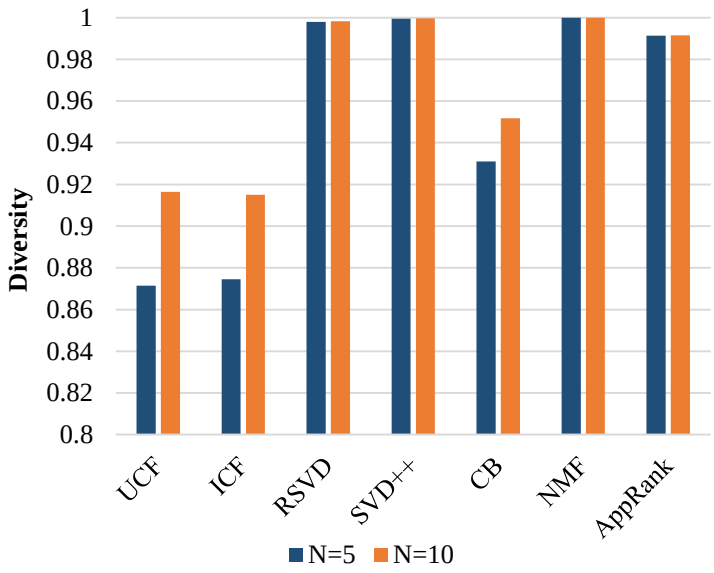


Figure 5.7. Comparison of Diversity
（Higher Diversity indicates better result）

Comparison between Free and Paid Apps. In this experiment, we split the dataset into two subsets. One subset only contains free apps and another only contains paid apps. We set *tp*=60% and compare Recall@100 of different methods on these two subsets as well as the whole dataset respectively. The results are shown in Figure 5.8. From the results, we find that for all methods, the recall values for both free app and paid app subsets are higher than for the whole dataset. This implies that users' interests and requirements are easier to predict within free apps and paid apps. Moreover, the recall values for the paid app subset are higher than for the free app subset. This is reasonable since users will consider more about what they need when they are installing paid apps, therefore it is easier to capture their requirements. On either the free app or paid app subset, our proposed AppRank method outperforms all the other methods.

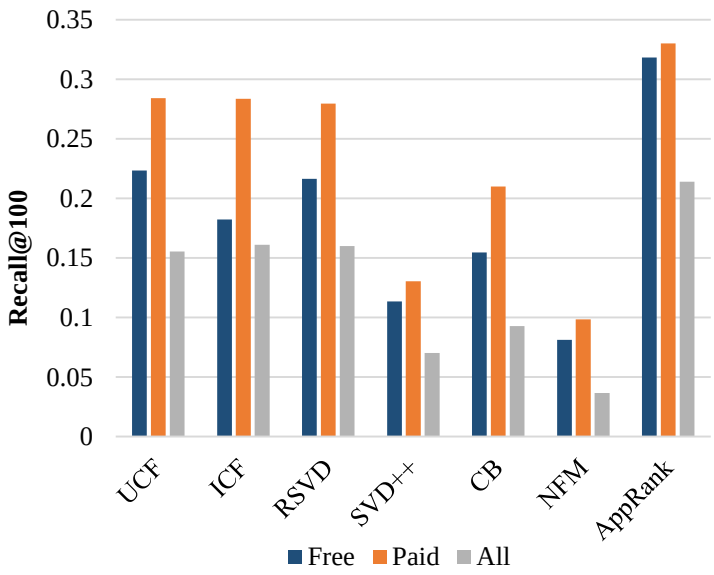


Figure 5.8. Comparison of Recall@100 for Free and Paid Subsets (Higher Diversity indicates better result)

5.6. Conclusion

In this study, we propose a functionality-based mobile app recommendation architecture. Our method recommends apps by revealing the detailed functionalities of apps and truly capturing users' functional requirements, which have not been considered by existing works. Furthermore, we prove that user reviews can be used to enrich item information and can be incorporated to enhance recommendation. The experiment conducted on a real-world dataset shows that our proposed AppRank method is effective in alleviating the data sparsity problem, and it is able to significantly improve recommendation accuracy and diversity.

Our work not only provides theoretical contributions to recommendation literature, but has practical implications as well. The proposed solution can be implemented as an effective real-world app recommender system helping users to discover apps that meet their requirements. The recommended apps would be more accurate, more diverse, and have less overlapping functionalities.

Our solution has some limitations. First, when ranking the candidate apps with similar functionalities, we simply use the apps' rating counts. In future work, it is possible to extract other non-functional aspects from user reviews, which can be incorporated in the ranking process to enable a personalized ranking approach. Second, as the rating data were collected from active users in

the evaluation, it may have some selection bias. This can be addressed in future work by collecting users' real usage data. Third, our method focuses more on the apps providing functionalities for users. However, there are also apps that may not be functionality-oriented, e.g. games. In future work, we will investigate the impact of product category on user requirement modeling, and extend our work by coming up with strategies to capture user requirements by differencing utilitarian and hedonic products.

CHAPTER 6. CONCLUSION

This thesis aims at addressing the data sparsity problem, which is one of the hardest problems affecting virtually all kinds of recommender systems. To achieve this goal, we propose to extract and incorporate meta-data from free-text User-Generated Content (UGC) into the recommendation process, seeking to make a difference to the quality, including accuracy, coverage, diversity and transparency of traditional recommendation algorithms.

This thesis consists of three different studies, each of which proposes a recommendation solution that incorporates UGC from different perspectives, and addresses specific problems introduced by data sparsity in different contexts. In particular, in study one, we show that adjective features embedded in user reviews are useful for characterizing item features as well as user tastes. In study two, we propose to model critic review articles at the topic level and use the inferred topics to represent item features and user interests. In study three, by extracting aspects from user reviews, we aim at building a mobile app recommendation solution that is able to model apps at the functional level and to recommend diverse mobile apps without redundancy.

There are several important contributions made by this thesis. First, it is proven in this thesis that UGC is a promising source for improving recommendation. Second, the adaptions of feature extraction techniques in this

thesis have implications for both UGC and RS research. Third, this thesis comes up with novel techniques to utilize textual content in the recommendation process, which fills the gap between UGC research and RS research.

This thesis also motivates several promising directions for future research. First, UGC is a valuable source for recommendation as well as many other applications. Beside the aspects used in this thesis, there are many types of information embedded in UGC that can be further explored. It is worthwhile to continue mining the value of UGC in future work. Second, cross-domain recommendation is still a challenging task in the present day. With the rapid growth of online review platforms, UGC is becoming increasingly available for most consumer products. It is interesting to see if UGC can act as a bridge to link different domains where no overlaps can be found in other dimensions, making cross-domain recommendation possible. Third, though it may appear that different strategies should be applied when recommending utilitarian versus hedonic products; however, in real-life systems, it is common that the same strategy is used in recommending the two types of products, because existing work might have difficulty in differentiating between them. In future work, it will be meaningful to explore how UGC can help to reveal the utilitarian and hedonic characteristics of products.

BIBLIOGRAPHY

Aciar, S., Zhang, D., Simoff, S., and Debenham, J. 2006. "Recommender System Based on Consumer Product Reviews," *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*: IEEE Computer Society, pp. 719-723.

Adomavicius, G., and Tuzhilin, A. 2005. "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Transactions on Knowledge and Data Engineering* (17:6), pp. 734-749.

Aggarwal, C.C., Wolf, J.L., Wu, K., and Yu, P.S. 1999. "Horting Hatches an Egg: A New Graph-Theoretic Approach to Collaborative Filtering," *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*: ACM, pp. 201-212.

Ahn, J.-w., Brusilovsky, P., Grady, J., He, D., and Syn, S.Y. 2007. "Open User Profiles for Adaptive News Systems: Help or Harm?," *Proceedings of the 16th International Conference on World Wide Web*: ACM, pp. 11-20.

Asuncion, A., Welling, M., Smyth, P., and Teh, Y.W. 2009. "On Smoothing and Inference for Topic Models," *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*: AUAI Press, pp. 27-34.

Böhmer, M., Bauer, G., and Krüger, A. 2010. "Exploring the Design Space of Context-Aware Recommender Systems That Suggest Mobile Applications," *2nd Workshop on Context-Aware Recommender Systems*.

Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. 2008. "Video Suggestion and Discovery for Youtube: Taking Random Walks through the View Graph," in: *Proceedings of the 17th International Conference on World Wide Web*. Beijing, China: ACM, pp. 895-904.

Bhandari, U., Sugiyama, K., Datta, A., and Jindal, R. 2013. "Serendipitous Recommendation for Mobile Apps Using Item-Item Similarity Graph," in *Information Retrieval Technology*. Springer, pp. 440-451.

Blei, D., Ng, A., and Jordan, M. 2003. "Latent Dirichlet Allocation," *Journal of Machine Learning Research* (3), pp. 993-1022.

Bonhard, P., and Sasse, M. 2006. "'Knowing Me, Knowing You'—Using Profiles and Social Networking to Improve Recommender Systems," *BT Technology Journal* (24:3), pp. 84-98.

Bouza, A., Reif, G., Bernstein, A., and Gall, H. 2008. "Semtree: Ontology-Based Decision Tree Algorithm for Recommender Systems," *International Semantic Web Conference*.

Cai, X., Bain, M., Krzywicki, A., Wobcke, W., Kim, Y., Compton, P., and Mahidadia, A. 2011. "Collaborative Filtering for People to People Recommendation in Social Networks " in *Ai 2010: Advances in Artificial Intelligence*, J. Li (ed.). Springer Berlin / Heidelberg, pp. 476-485.

Cai, Y., Leung, H.-f., Li, Q., Min, H., Tang, J., and Li, J. 2014. "Typicality-Based Collaborative Filtering Recommendation," *IEEE Transactions on Knowledge and Data Engineering* (26:3), pp. 766-779.

Celma, O., Ramírez, M., and Herrera, P. 2005. "Foafing the Music: A Music Recommendation System Based on Rss Feeds and User Preferences," *in ISMIR*: Citeseer.

Chen, L., and Wang, F. 2013. "Preference-Based Clustering Reviews for Augmenting E-Commerce Recommendation," *Knowledge-Based Systems* (50), Sep, pp. 44-59.

Christakou, C., Vrettos, S., and Stafylopatis, A. 2007. "A Hybrid Movie Recommender System Based on Neural Networks," *International Journal on Artificial Intelligence Tools* (16:05), pp. 771-792.

Clever, N., Kirchner, A., Schray, D., and Schulte, M. 2009. "User-Generated Content," in: *Essay, Institut für Wirtschaftsinformatik. Westfälische Wilhelms-universität, Münster*. pp. 1-3.

Cohen, J. 1995. "Highlights: Language- and Domain-Independent Automatic Indexing Terms for Abstracting," *Journal of the American Society for Information Science* (46:3), pp. 162-174.

Datta, A., Dutta, K., Kajanan, S., and Pervin, N. 2012. "Mobilewalla: A Mobile Application Search Engine," in *Mobile Computing, Applications, and Services*. Springer, pp. 172-187.

de Gemmis, M., Lops, P., Semeraro, G., and Basile, P. 2008. "Integrating Tags in a Semantic Content-Based Recommender," *Proceedings of the 2008 ACM conference on Recommender systems*, Lausanne, Switzerland: ACM, pp. 163-170.

Deerwester, S., Dumais, S., Landauer, T., Furnas, G., and Harshman, R. 1990. "Indexing by Latent Semantic Analysis," *Journal of the American Society of Information Science* (41:6), pp. 391-407.

Deshpande, M., and Karypis, G. 2004. "Item-Based Top-N Recommendation Algorithms," *ACM Transactions on Information Systems (TOIS)* (22:1), pp.

143-177.

Diederich, J., and Iofciu, T. 2006. "Finding Communities of Practice from User Profiles Based on Folksonomies," *Innovative Approaches for Learning and Knowledge Sharing, EC-TEL Workshop Proc*, pp. 288-297.

Funk, S. 2006. "Netflix Update: Try This at Home." from http://sifter.org/simon/journal/20061211.html

Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S., and Schmidt-Thieme, L. 2010. "Learning Attribute-to-Feature Mappings for Cold-Start Recommendations," *Data Mining (ICDM), 2010 IEEE 10th International Conference on*: IEEE, pp. 176-185.

Ganu, G., Kakodkar, Y., and Marian, A. 2013. "Improving the Quality of Predictions Using Textual Information in Online User Reviews," *Information Systems* (38:1), pp. 1-15.

Golbeck, J. 2006. "Generating Predictive Movie Recommendations from Trust in Social Networks," in: *Trust Management*. Springer, pp. 93-104.

Groh, G., Birnkammerer, S., and Köllhofer, V. 2012. "Social Recommender Systems," in *Recommender Systems for the Social Web*. Springer Berlin Heidelberg, pp. 3-42.

Gutta, S., Kurapati, K., Lee, K., Martino, J., Milanski, J., Schaffer, J.D., and Zimmerman, J. 2000. "Tv Content Recommender System," *Proceedings of the National Conference on Artificial Intelligence*: Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, pp. 1121-1122.

Harb, A., Planti é, M., Dray, G., Roche, M., Trousset, F., and Poncelet, P. 2008. "Web Opinion Mining: How to Extract Opinions from Blogs?," *Proceedings of the 5th International Conference on Soft Computing as Transdisciplinary Science and Technology*: ACM, pp. 211-217.

Herlocker, J.L., Konstan, J.A., Terveen, L.G., and Riedl, J.T. 2004. "Evaluating Collaborative Filtering Recommender Systems," *ACM Transactions on Information Systems (TOIS)* (22:1), pp. 5-53.

Hofmann, T. 2004. "Latent Semantic Models for Collaborative Filtering," *ACM Transactions on Information System* (22:1), pp. 89-115.

Hotho, A., J äschke, R., Schmitz, C., Stumme, G., and Althoff, K.-D. 2006. "Folkrank: A Ranking Algorithm for Folksonomies," *LWA*, pp. 111-114.

Hsieh, C.-J., and Dhillon, I.S. 2011. "Fast Coordinate Descent Methods with Variable Selection for Non-Negative Matrix Factorization," *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery*

and Data Mining: ACM, pp. 1064-1072.

Huang, Z., Chen, H., and Zeng, D. 2004. "Applying Associative Retrieval Techniques to Alleviate the Sparsity Problem in Collaborative Filtering," *ACM Transactions on Information Systems (TOIS)* (22:1), pp. 116-142.

Järvelin, K., and Kekäläinen, J. 2002. "Cumulated Gain-Based Evaluation of Ir Techniques," *ACM Transactions on Information Systems (TOIS)* (20:4), pp. 422-446.

Jakob, N., Weber, S.H., M, M.C., and Gurevych, I. 2009. "Beyond the Stars: Exploiting Free-Text User Reviews to Improve the Accuracy of Movie Recommendations," in: *Proceedings of the 1st International CIKM Workshop on Topic-sentiment Analysis for Mass Opinion*. Hong Kong, China: ACM, pp. 57-64.

Jamali, M., and Ester, M. 2010. "A Matrix Factorization Technique with Trust Propagation for Recommendation in Social Networks," in: *Proceedings of the Fourth ACM Conference on Recommender Systems*. Barcelona, Spain: ACM, pp. 135-142.

Jeh, G., and Widom, J. 2003. "Scaling Personalized Web Search," *Proceedings of the 12th International Conference on World Wide Web*: ACM, pp. 271-279.

Karatzoglou, A., Baltrunas, L., Church, K., and Böhmer, M. 2012. "Climbing the App Wall: Enabling Mobile App Discovery through Context-Aware Recommendations," *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*: ACM, pp. 2527-2530.

Katz, G., Ofek, N., Shapira, B., Rokach, L., and Shani, G. 2011. "Using Wikipedia to Boost Collaborative Filtering Techniques," *Proceedings of the Fifth ACM Conference on Recommender Systems*: ACM, pp. 285-288.

Kohlschütter, C., Fankhauser, P., and Nejdl, W. 2010. "Boilerplate Detection Using Shallow Text Features," *Proceedings of the Third ACM International Conference on Web Search and Data Mining*: ACM, pp. 441-450.

Koren, Y. 2008. "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model," *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, Nevada, USA: ACM, pp. 426-434.

Koren, Y., and Bell, R. 2011. "Advances in Collaborative Filtering," in *Recommender Systems Handbook*, F. Ricci, L. Rokach and B. Shapira (eds.). Springer, pp. 145-186.

Kullback, S. 1987. "The Kullback-Leibler Distance," *The American Statistician* (41:4), pp. 340-341.

Lan, M., Tan, C.L., Su, J., and Lu, Y. 2009. "Supervised and Traditional Term Weighting Methods for Automatic Text Categorization," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (31:4), pp. 721-735.

Lee, D.D., and Seung, H.S. 1999. "Learning the Parts of Objects by Non-Negative Matrix Factorization," *Nature* (401:6755), pp. 788-791.

Lemire, D., and Maclachlan, A. 2005. "Slope One Predictors for Online Rating-Based Collaborative Filtering," *SDM*: SIAM, pp. 1-5.

Li, B., Yang, Q., and Xue, X. 2009. "Can Movies and Books Collaborate? Cross-Domain Collaborative Filtering for Sparsity Reduction," *Proceedings of the 21st International Jont Conference on Artifical Intelligence*: Morgan Kaufmann Publishers Inc., pp. 2052-2057.

Lin, C.-J. 2007. "Projected Gradient Methods for Nonnegative Matrix Factorization," *Neural Computation* (19:10), pp. 2756-2779.

Lin, J., Sugiyama, K., Kan, M.-Y., and Chua, T.-S. 2013. "Addressing Cold-Start in App Recommendation: Latent User Models Constructed from Twitter Followers," *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*: ACM, pp. 283-292.

Liu, Q., Ma, H., Chen, E., and Xiong, H. 2013. "A Survey of Context-Aware Mobile Recommendations," *International Journal of Information Technology & Decision Making* (12:01), pp. 139-172.

Lops, P., Gemmis, M., and Semeraro, G. 2011. "Content-Based Recommender Systems: State of the Art and Trends," in *Recommender Systems Handbook*, F. Ricci, L. Rokach and B. Shapira (eds.). Springer, pp. 73-105.

Maneeroj, S., and Takasu, A. 2009. "Hybrid Recommender System Using Latent Features," *International Conference on Advanced Information Networking and Applications Workshops*: IEEE, pp. 661-666.

Manzato, M.G. 2012. "Discovering Latent Factors from Movies Genres for Enhanced Recommendation," *Proceedings of the Sixth ACM Conference on Recommender Systems*: ACM, pp. 249-252.

McNee, S.M., Riedl, J., and Konstan, J.A. 2006. "Being Accurate Is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems," *CHI'06 Extended Bbstracts on Human Factors in Computing Systems*: ACM, pp. 1097-1101.

Michlmayr, E. 2007. "Learning User Profiles from Tagging Data and Leveraging Them for Personal(Ized) Information Access," *In Proceedings of the Workshop on Tagging and Metadata for Social Information Organization, 16th International World Wide Web Conference*.

Middleton, S.E., Shadbolt, N.R., and De Roure, D.C. 2004. "Ontological User Profiling in Recommender Systems," *ACM Transactions on Information Systems (TOIS)* (22:1), pp. 54-88.

Mihalcea, R., and Tarau, P. 2004. "Textrank: Bringing Order into Texts," *Conference on Empirical Methods in Natural Language Processing*

Page, L., Brin, S., Motwani, R., and Winograd, T. 1999. "The Pagerank Citation Ranking: Bringing Order to the Web," *Stanford Digital Library Technologies Project*).

Paterek, A. 2007. "Improving Regularized Singular Value Decomposition for Collaborative Filtering," *Proceedings of KDD Cup and Workshop*, pp. 5-8.

Pazzani, M.J., and Billsus, D. 2007. "Content-Based Recommendation Systems," in *The Adaptive Web*. Springer, pp. 325-341.

Pucci, A., Gori, M., and Maggini, M. 2007. "A Random-Walk Based Scoring Algorithm Applied to Recommender Engines," in *Advances in Web Mining and Web Usage Analysis*. Philadelphia, PA, USA: Springer Berlin Heidelberg, pp. 127-146.

Rafailidis, D., Axenopoulos, A., Etzold, J., Manolopoulou, S., and Daras, P. 2014. "Content-Based Tag Propagation and Tensor Factorization for Personalized Item Recommendation Based on Social Tagging," *ACM Transactions on Interactive Intelligent Systems (TiiS)* (3:4), p. 26.

Ramage, D., Hall, D., Nallapati, R., and Manning, C.D. 2009. "Labeled Lda: A Supervised Topic Model for Credit Attribution in Multi-Labeled Corpora," *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*: Association for Computational Linguistics, pp. 248-256.

Ramage, D., Manning, C.D., and Dumais, S. 2011. "Partially Labeled Topic Models for Interpretable Text Mining," *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*: ACM, pp. 457-465.

Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., and Riedl, J. 1994. "Grouplens: An Open Architecture for Collaborative Filtering of Netnews," *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*: ACM, pp. 175-186.

Said, A., De Luca, E.W., and Albayrak, S. 2010. "How Social Relationships Affect User Similarities," *Proceedings of the Workshop on Social Recommender Systems*.

Salakhutdinov, R., and Mnih, A. 2008. "Probabilistic Matrix Factorization," *Advances in Neural Information Processing Systems* (20), pp. 1257-1264.

Salakhutdinov, R., Mnih, A., and Hinton, G. 2007. "Restricted Boltzmann Machines for Collaborative Filtering," *Proceedings of the 24th International Conference on Machine Learning*, Corvalis, Oregon: ACM, pp. 791-798.

Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. 2001. "Item-Based Collaborative Filtering Recommendation Algorithms," *Proceedings of the 10th International Conference on World Wide Web*, pp. 285-295.

Schein, A.I., Popescul, A., Ungar, L.H., and Pennock, D.M. 2002. "Methods and Metrics for Cold-Start Recommendations," *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*: ACM, pp. 253-260.

Schwab, I., Kobsa, A., and Koychev, I. 2001. "Learning User Interests through Positive Examples Using Content Analysis and Collaborative Filtering," *Internal Memo, GMD, St. Augustin, Germany*).

Semeraro, G., Basile, P., de Gemmis, M., and Lops, P. 2009. "User Profiles for Personalizing Digital Libraries."

Sen, S., Vig, J., and Riedl, J. 2009. "Tagommenders: Connecting Users to Items through Tags," in: *Proceedings of the 18th International Conference on World wide web*. Madrid, Spain: ACM, pp. 671-680.

Seung, D., and Lee, L. 2001. "Algorithms for Non-Negative Matrix Factorization," *Advances in Neural Information Processing Systems* (13), pp. 556-562.

Shani, G., and Gunawardana, A. 2011. "Evaluating Recommendation Systems," in *Recommender Systems Handbook*, F. Ricci, L. Rokach and B. Shapira (eds.). Springer, pp. 73-105.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Hanjalic, A., and Oliver, N. 2012. "Tfmap: Optimizing Map for Top-N Context-Aware Recommendation," *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*: ACM, pp. 155-164.

Spaeth, A., and Desmarais, M.C. 2013. "Combining Collaborative Filtering and Text Similarity for Expert Profile Recommendations in Social Websites,"

Proceedings of The 21st Conference on User Modeling, Adaptation and Personalization, Rome, Italy.

Srebro, N., and Jaakkola, T. 2003. "Weighted Low-Rank Approximations," *Proceedings of the 20th International Conference on Machine Learning*, pp. 720-727.

Su, X., and Khoshgoftaar, T.M. 2009. "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence* (2009), p. 4.

Sueiras, J., Salafranca, A., and Florez, J.L. 2007. "A Classical Predictive Modeling Approach for Task Who Rated What? Of the Kdd Cup 2007," *ACM SIGKDD Explorations Newsletter* (9:2), pp. 57-61.

Toutanova, K., Klein, D., Manning, C.D., and Singer, Y. 2003. "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network," in: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*. Edmonton, Canada: Association for Computational Linguistics, pp. 173-180.

Voll, K., and Taboada, M. 2007. "Not All Words Are Created Equal: Extracting Semantic Orientation as a Function of Adjective Relevance," in *Ai 2007: Advances in Artificial Intelligence*. Springer, pp. 337-346.

Wang, Z., Tan, Y., and Zhang, M. 2010. "Graph-Based Recommendation on Social Networks," in: *Proceedings of the 2010 12th International Asia-Pacific Web Conference*. IEEE Computer Society, pp. 116-122.

Wei, C., Hsu, W., and Lee, M. 2011. "A Unified Framework for Recommendations Based on Quaternary Semantic Analysis," *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Beijing, China: ACM, pp. 1023-1032.

Xia, X., Wang, X., Zhou, X., and Zhu, T. 2014. "Collaborative Recommendation of Mobile Apps: A Swarm Intelligence Method," in *Mobile, Ubiquitous, and Intelligent Computing*. Springer, pp. 405-412.

Xin, X., Lyu, M.R., and King, I. 2011. "Cmap: Effective Fusion of Quality and Relevance for Multi-Criteria Recommendation," *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*: ACM, pp. 455-464.

Xu, J.A., and Araki, K. 2006. "A Svm-Based Personal Recommendation System for Tv Programs," *Proceedings of the 12th International Multi-Media Modelling Conference*: IEEE, p. 4 pp.

Xue, G.-R., Lin, C., Yang, Q., Xi, W., Zeng, H.-J., Yu, Y., and Chen, Z. 2005. "Scalable Collaborative Filtering Using Cluster-Based Smoothing," *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*: ACM, pp. 114-121.

Yan, B., and Chen, G. 2011. "Appjoy: Personalized Mobile Application Discovery," *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*: ACM, pp. 113-126.