MOBILE APP RECOMMENDATION

JOVIAN LIN

(B.Comp. (Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2014

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.

JOVIAN LIN

20 JUNE 2014

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support, direction, and love of a multitude of people.

First, I would like to express my deepest gratitude to my PhD advisors, Prof. Tat-Seng Chua and A/P Min-Yen Kan, for their steadfast support and intellectual guidance. There have been countless occasions when I have felt hopelessly lost, disheartened and stumped about the direction of my research. But inevitably (and *thankfully*), a meeting with them would reinvigorate my enthusiasm and lift my spirits — and most importantly, guide me back in the right direction.

I would like to thank Dr. Kazunari Sugiyama for his meticulous proofreading and valuable suggestions. His thorough attention to detail has helped me to spot the most obscure mistakes, leading to better quality works. I would also like to thank Dr. Zhaoyan Ming for her invaluable guidance during the start of my PhD. Her patience and encouragement has helped me overcome the despair that I have felt during that period.

I am grateful to the members of my thesis committee, Prof. Chew-Lim Tan, Prof. Mong-Li Lee, A/P Yi Zhang, A/P Anindya Datta, and A/P Ye Wang, for their critical reading of the thesis and providing their valuable advice, which have helped me further improve this thesis.

I have also been blessed to have had many supporting my endeavors since the beginning of my PhD journey, playing multiple roles for which I am greatly thankful for:

My advisors from NUS Enterprise, Prof. Juzar Motiwalla and Dr. Pete Kellock, for guiding me down the entrepreneurial path and whetting my appetite for it, as well as Masana Takashi for inspiring me with his unwavering optimism and positivity;

My colleagues from the Web IR/NLP Group (WING): Jun-Ping Ng, Aobo Wang, Tao Chen, Xiangnan He, and Muthu Chandrasekaran;

My colleagues from the Lab for Media Search (LMS), both past and present: Shiyong Neo, Yantao Zheng, Guangda Li, Xiaojian Zhao, Zhe Chen, Liqiang Nie, Hanwang Zhang, Yiliang Zhao, Yan Chen, Jingwen Bian, and Xue Geng;

Dr. James Wong for surgically repairing my collapsed lung (or pneumothorax) — which I was diagnosed with just 10 days before I was to attend my first Rank 1 conference overseas — and allowing me to proceed on with SIGIR'13 with minimal health risk;

My friends Wen-Shih Wee, Madankumar Balakrishnan, Dillion Tan, Kangli Yip, Kah-Ming Tan, Zhanwei Lim, Yawsing Tan, Gabriel Leong, Stephan Hassold, Christine Chong, Lionel Chan, Jasper Fay, Jean Hair, Xuhui Chan, Hannah Watson, Fiona Lim, and countless others for showing love and support in both times of great happiness and deep depression;

and most importantly, my parents and two sisters, Erinna and Elisa, for their understanding and support throughout these years.

CONTENTS

ABSTRACT

Mobile apps have become commonplace in society. But with millions of apps flooding the app stores, recommender systems have become indispensable tools as they help consumers overcome the problem of *information overload*. By sifting through the ocean of apps, they allow consumers to discover new and compelling apps through personalized recommendations. Yet, conventional recommender systems have their own set of problems — particularly the problem of data sparsity, which is the result of insufficient ratings per app. Furthermore, conventional recommender systems do not account for the singularity of the app domain that, if properly utilized, could potentially provide significant improvements to current app recommender systems.

In this thesis, we investigate the singularity of the app domain for the purpose of improving app recommendations. By exploiting the app domain's unique characteristics, we come up with novel recommendation techniques that take advantage of information from social networks, version updates, and a slew of app metadata that is typically underused.

First, we describe an approach that accounts for nascent information culled from Twitter to provide relevant recommendations in cold-start situations. By exploiting an app's Twitter handle (*e.g.*, @angrybirds), we extract its Twitter-followers and show how these Twitter-followers can act as an alternative source of information to overcome the cold-start problem.

Second, we observe that in the domain of mobile apps, a *version update* may provide substantial changes to an app which may revive a consumer's interest for a previously unappealing version. We leverage version features for the purpose of improving app recommendations, and show that incorporating version information into conventional techniques significantly improves the recommendation quality.

Finally, given a diverse set of app recommendation techniques, we propose a unifying framework that marries the strengths of the various individual techniques while overcoming their respective weaknesses. We present a hybrid app recommender system that utilizes both conventional and novel app recommendation techniques — as well as the assimilation of user and app metadata features — for the purpose of generating a personalized ranked list of recommended apps.

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This chapter bootstraps the thesis by explaining the problem we are attempting to solve. We also summarize the key contributions that we have achieved here.

With an ever-increasing number of smartphones and tablets entering the consumer marketplace, mobile devices have become an indispensable part of our daily lives. Because of this growth in the mobile device market, mobile applications (or "apps" in short) are also on the rise and ever in demand[1,2] — as the heart of mobile devices lies in the apps. Furthermore, as important as apps are to their users, they are even more so for enterprises. Among other things, apps have revolutionized consumer behavior and changed the way in which they shop, making it crucial for enterprises to tap into the mobile app market as well.

[1] "Apple's App Store Marks Historic 50 Billionth Download," Apple Press Info, accessed on Sep 10, 2013, `http://www.apple.com/sg/pr/library/2013/05/16Apples-App-Store-Marks-Historic-50-Billionth-Download.html`.

[2] "Google Play Now Generates More Downloads than iOS App Store," Forbes, accessed on Sep 10, 2013, `http://www.forbes.com/sites/terokuittinen/2013/07/31/google-play-now-generates-more-downloads-than-ios-app-store/`.

The reasons above have led to an explosive growth of app stores[3,4,5], launching a gigantic explosion of consumer interest in the mobile field that creates economic opportunities for app developers, companies, and marketers. While this growth has provided users with a myriad of unique and useful apps, the sheer number of choices also makes it more difficult for users to find apps that are relevant to their interests. In other words, app stores face the problem of *information overload* whereby consumers experience difficulty in finding relevant apps.

To alleviate the problem of information overload, recommender systems have been deployed in app stores to provide personalized recommendations for users. Existing recommender systems typically focus on the following techniques: *i*) collaborative filtering, which works by recommending items (*i.e.*, apps) to target users based on what other similar users have previously preferred; and *ii*) content-based filtering, which provides recommendations by comparing representations of the content of an item against what the target user is interested in.

Unfortunately, as collaborative filtering depends on ratings to generate recommendations, a common problem is that a new app that has no prior ratings cannot be recommended; at least not until more users provide ratings for it. This is widely known as the *cold-start* problem, and it plagues the app store because many excellent apps do not have enough ratings, causing them to go unnoticed[6]. The only way for recommender systems to provide automatic recommendation is to either wait for sufficient ratings to be supplied by users — which will take some time — or rely

[3] `https://www.apple.com/itunes/`

[4] `https://play.google.com/store`

[5] `https://www.windowsphone.com/en-us/store/`

[6] Too many good apps are victims of their own invisibility, lying buried, unused, and unknown. Due to the lack of visibility, many apps do not have sufficient ratings, which hinders the use of collaborative filtering techniques. Furthermore, popular apps remain popular while other (possibly superior) apps stay buried in the app store; in fact, more than 50% of apps have 10 or fewer ratings, and over 30% have too few ratings for any to be reported.

on content-based filtering. However, as content-based filtering algorithms seek to recommend items based on similar content, an obvious drawback is that the recommended items are similar to the user's previously-consumed items; in other words, there is a lack of diversity in the recommendations generated by content-based filtering (Park and Chu, 2009). For example, if a consumer has only downloaded weather-related apps, content-based filtering would only recommend other weather-related apps. This lack of diversity results in unsatisfactory recommendations.

1.1 Motivation

Conventional recommender systems do not account for the singularity of the app domain that, if properly utilized, could provide significant improvements to current state-of-the-art recommendation techniques. In this section, we present an overview of the unique characteristics of the app domain and explain how, by exploiting distinctive features in the app domain, we can overcome the cold-start problem as well as improve the recommendation quality.

1.1.1 Nascent Signals from Microblogs

With the rise of social networking services, people broadcast to and message friends, colleagues, and the general public about many subjects — including the topic of apps. An interesting possibility thus arises: Can we merge information mined from the rich data or nascent signals in social networks to enhance the performance of app recommendation? Through our case studies, we verified that the answer to this question is indeed "yes." Figure 1.1 illustrates a case study of our observation with the "Evernote" app. It was released in May 2012 and had no ratings in the iTunes App Store for two months; it was only in July 2012 that the first few ratings

Figure 1.1: Timeline of the "Evernote" app.

started coming in. However, by May 2012, Evernote's Twitter account already had more than 120,000 followers and 1,300 tweets. Given this encouraging observation, one of our works takes advantage of this active yet indirect information that is present in Twitter and use it to alleviate the cold-start problem that besets newly released apps.

1.1.2 Apps Contain Various Versions

We observe that existing recommender systems[7] usually model items as static — unchanging in attributes, description, and features. However, apps are different, for they change and evolve with every revision (illustrated in Figure 1.2). Hence, an app that was unpopular in the past may become popular after a version update. For example, Version 1.0 of App X did not interest a user at first, but a recent update to Version 2.0 — which promises to provide the functionality of high definition (HD) video capture — may arouse his interest in the revised app. A conventional recommender system that regards an app as static would fail to capture this important detail. This is why it is vital for app recommender systems to process nascent signals in version descriptions to identify desired functionalities that users are looking for. Furthermore, version descriptions constitute an important recommendation evidence source as well as a basis for understanding the general rationale for a recommendation.

[7]Conventional items in typical recommender systems include books, music, movies, and points-of-interests (*i.e.*, locations).

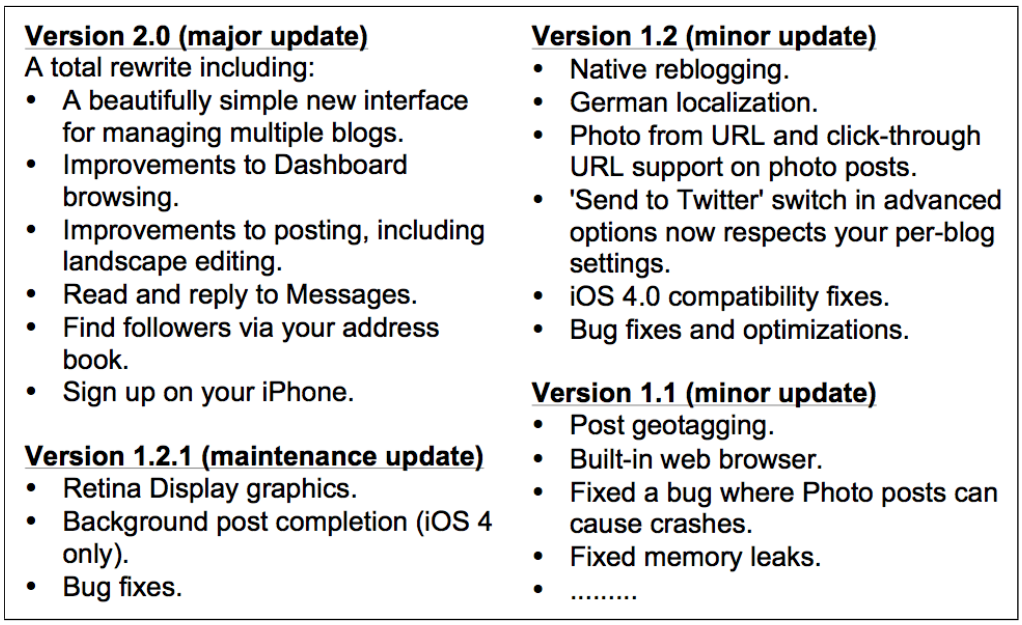


Figure 1.2: An app's changelog chronicles the details of every version update.

1.1.3 The Unifying Framework

A variety of recommendation techniques have been proposed as the basis for recommender systems: collaborative filtering, content-based filtering, as well as the aforementioned techniques that utilize information from Twitter and version features. Each of these techniques has well-known shortcomings, such as being affected by the cold-start problem or the lack of a Twitter handle. A hybrid recommender system is an approach that combines multiple techniques together to achieve some synergy between them. For example, collaborative filtering and content-based filtering might be combined so that the content-based component can compensate for the cold-start problem that plagues collaborative filtering. Besides, as observed in BellKor's winning entry of the Netflix Prize (Koren, 2009), the more techniques we combine, the more robust will the recommender system be. Therefore, our final work examines a unifying framework that marries the strengths of the various individual techniques while overcoming their re-

spective weaknesses; not only does the unifying framework combine the outputs of the individual recommendation techniques, it also assimilates the user and app metadata features. Interestingly, the results of our analysis coincides with the findings from consumer analytics.

1.2 Contributions of the Thesis

This thesis makes the following contributions in the area of app recommender systems. They are summarized as follows:

1. **Using nascent signals in microblogs to alleviate the cold-start problem in mobile app recommendation**. We describe a method that accounts for nascent information culled from Twitter to provide relevant recommendation in cold-start situations. We use Twitter handles to access an app's Twitter account and extract the IDs of their Twitter-followers. We create *pseudo-documents* that contain the IDs of Twitter users interested in an app and then apply latent Dirichlet allocation (LDA) to generate latent groups. At test time, a target user seeking recommendations is mapped to these latent groups. By using the transitive relationship of latent groups to apps, we estimate the probability of the user liking the app. We show that by incorporating information from Twitter, our approach overcomes the difficulty of cold-start app recommendation and significantly outperforms other state-of-the-art recommendation techniques in this situation.

2. **Using version features in mobile app recommendation**. We present a novel framework that incorporates features distilled from version descriptions into app recommendation. We utilized a semi-supervised topic model to construct a representation of an app's ver-

sion as a set of latent topics from version metadata and textual descriptions. We then discriminate the topics based on genre information and weight them on a per-user basis to generate a version-sensitive ranked list of apps for a target user. Incorporating our version features with state-of-the-art individual and hybrid recommendation techniques significantly improves recommendation quality. An important advantage of our method is that it targets particular versions of apps, allowing previously disfavored apps to be recommended when user-relevant features are added.

3. **A unifying framework that integrates conventional recommendation techniques, state-of-the-art app recommendation techniques, as well as user and app metadata features**. Because different recommendation techniques work in different scenarios, we present a framework to integrate the various sources of information — from the output scores of various recommendation techniques to the user and app metadata features — into a hybrid model that is able to recommend a set of apps to a target user. This hybrid model employs gradient tree boosting (GTB) (Friedman, 2001) to integrate the aforementioned features, and the unifying framework combines the strengths of individual recommendation techniques to overcome their individual shortcomings.

1.2.1 Research Publications

The work in this thesis has been published in the following conferences:

- Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. *Addressing Cold-Start in App Recommendation: Latent User Models Constructed from Twitter Followers.* In Proceedings of the 36th Annual International ACM SIGIR Conference on Research and Devel-

opment in Information Retrieval (SIGIR'13), pages 283–292, Dublin, Ireland, July 28–August 1, 2013.

- Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. *New and Improved: Modeling Versions to Improve App Recommendation.* In Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'14), pages 647–656, Gold Coast, Australia, July 6–11, 2014.

1.3 Outline of the Thesis

This thesis is structured into 6 chapters.

- **Chapter 2** discusses the background and related work for this thesis in the following five areas: *i*) collaborative filtering, *ii*) content-based filtering, *iii*) social-based recommendation, *iv*) hybrid recommender systems, and *v*) recommender systems that are applied in the domain of mobile apps.

- **Chapter 3** describes how nascent signals in microblogs — particularly Twitter-followers — can be used in app recommendation. This work combines information from the domains of apps and Twitter to alleviate the cold-start problem.

- **Chapter 4** describes how version features (which are unique in the app domain) can be used to enhance recommendation accuracy. We show that version descriptions can be an alternative to noisy app descriptions. We present a method that uses a semi-supervised variant of latent Dirichlet allocation (LDA) to build this recommendation technique.

- **Chapter 5** unifies all the recommendation techniques — conventional as well as novel — into a hybrid app recommender system. We show that by including user and app metadata features with the individual recommendation scores that are generated from the various recommendation techniques, we can achieve significant improvements to individual and hybrid baselines.

- Finally, **Chapter 6** summarizes the works in this thesis and outlines a number of future directions.

Chapter 2

Background

Before detailing the work that we have done, this chapter provides the necessary background knowledge.

In this chapter, we review key background information on recommender systems. The major directions of recommender systems can be categorized into four main streams: *i*) collaborative filtering, *ii*) content-based filtering, *iii*) social-based filtering, and *iv*) hybrid recommender systems (see Sections 2.1, 2.2, 2.3, and 2.4, respectively).

Collaborative filtering employs the rating history of users, whereas content-based filtering utilizes the content features of the apps. Social-based filtering takes advantage of a user's social network (*e.g.*, Facebook, Twitter, etc.) to recommend items that the user's friends have interacted with. We first describe the methods relating to the three aforementioned systems, followed by a background review of the concepts in hybrid recommender systems. Finally, we discuss a number of notable recent works on mobile app recommendation.

2.1 Collaborative Filtering

Collaborative filtering is a well-known recommendation technique that has been widely adopted and studied. The fundamental assumption of collaborative filtering is that if two users rated n items similarly, or have similar behaviors (*e.g.*, buying, watching, listening, "liking"), they will in turn rate other items similarly (Goldberg et al., 2000). Collaborative filtering techniques use a database of preferences for items by users to predict additional items a user might like. In a typical scenario, there is a list of m users $\{u_1, u_2, \ldots, u_m\}$ and a list of n items $\{i_1, i_2, \ldots, i_n\}$, and each user, u_i, has a list of items, I_{u_i}, which the user has rated, or about which their preferences have been inferred through their behaviors (Su and Khoshgoftaar, 2009). The ratings can either be explicit indications, such as the 5-point Likert scale, or implicit indications, such as purchases or click-throughs (Miller et al., 2004). Collaborative filtering represents the most popular recommendation technique due to its compelling simplicity and excellent quality of recommendations (Ziegler et al., 2005). In addition, collaborative filtering algorithms can be further divided into memory-based and model-based approaches (Cremonesi et al., 2010).

2.1.1 Memory-based Collaborative Filtering

In a memory-based approach, a recommendation is made by determining the nearest neighbors of a user and/or an app, and then aggregating the ratings of these neighbors. Memory-based techniques have the advantage of being better adapted to users with unusual tastes, but they are impractical due to scalability issues since calculating the neighborhood for users and items can be time consuming, especially in real life or commercial datasets (Hofmann, 2003). Notable examples of memory-based collaborative filtering systems include GroupLens (Resnick et al., 1994) as well

as Amazon[1] (Linden et al., 2003). Memory-based approaches can be further classified into two types: i) user-based (Resnick et al., 1994) and ii) item-based (Sarwar et al., 2001).

i) **User-based Approach**. The intuition behind user-based collaborative filtering is that a user would be interested in the items that are also liked by other users who share the same tastes with him or her. The basic idea is to first calculate a similarity score, $w_{u,v}$, between user u and user v based on their ratings of similar items. Cosine-similarity is often used in this case. After which, based on the k most similar users, a set of items, C, is extracted based on the frequency of the items and the top-N most frequent items in C (that the target user has not consumed) is recommended.

ii) **Item-based Approach**. The item-based approach (Linden et al., 2003; Sarwar et al., 2001) became popular later. The intuition behind item-based collaborative filtering is that the users would be interested in the items that are similar to those he or she liked in the past. The basic idea is to first calculate a similarity score, $w_{i,j}$, between item i and item j based on the users who have rated both of these items. After which, the similarity score is used to predict which items should be recommended to the target user, based on the user's previously consumed items.

Despite their popularity, the memory-based approaches suffer from the problem of data sparsity in which the number of ratings obtained is very small compared to the number of ratings needed needed for prediction (Adomavicius and Tuzhilin, 2005). To help diminish the effects of data sparsity, model-based collaborative filtering has been investigated.

[1] `https://www.amazon.com`

2.1.2 Model-based Collaborative Filtering

Model-based techniques learn to recognize complex patterns through training data, and then use the trained models to make predictions for recommendation tasks. Compared to memory-based techniques, model-based techniques are better at addressing the data sparsity problem and improving the prediction performance. Typically, model-based collaborative filtering are represented by regression models, classification models, and latent factor models.

i) **Regression Models**. Regression estimates how the typical value of the dependent variable changes when any one of the independent variables is changed. It is a simple and effective method to make predictions for numerical values of users' preferences, such as the 5-point Likert scale ratings or binary liked/disliked values. Vucetic and Obradovic (2005) proposed a regression-based collaborative filtering method that builds a collection of simple linear models by searching for similarities between items, and combines them to effectively provide rating predictions for a target user; whereas Lemire and Maclachlan (2005) proposed three Slope One schemes to estimate the average difference between the ratings of one item and another for users who rated both.

ii) **Classification Models**. For recommender systems in which user ratings are categorical (*e.g., liked* or *disliked*), recommendation can be regarded as a classification problem. Miyahara and Pazzani (2000) proposed a simple Bayesian collaborative filtering method that employs a strategy based on Naive Bayes to perform the collaborative filtering task. They assumed that other users' preferences regarding a target item are independent from the target user's preference on the same item. After training the Bayesian model, the probability

that the target user will like the target item can be computed given the other users' preferences on the same item. Su and Khoshgoftaar (2006) extended the simple Bayesian collaborative filtering method by applying a more advanced model — the Bayesian Belief Network — that addresses the data sparsity problem and is able to handle data that is multi-class. Other notable classification models include Cho et al. (2002) and Nikovski and Kulev (2006) where they combined decision trees with association rules and applied standard tree-learning algorithms to simplify the recommendation rules.

iii) **Latent Factor Models**. Latent factor models such as probabilistic matrix factorization (PMF) comprise of an alternative approach to collaborative filtering by transforming both items and users into the same latent factor space. The more popular and successful latent factor models are based on the concept of dimensionality reduction that aims to provide the best lower rank approximations of the original user-item ratings matrix. Notable techniques include probabilistic semantic analysis (PLSA) (Hofmann, 2004), principal component analysis (PCA) (Kim and Yum, 2005), restricted Boltzmann machine (RBM) (Salakhutdinov et al., 2007), and singular vector decomposition (SVD) (Takács et al., 2008). These techniques deal better with data sparsity and have gained immense popularity due to their accuracy and scalability.

2.1.3 Graph-based Collaborative Filtering

Graph-based collaborative filtering represents data as a graph in which users and items are represented as nodes, while edges capture the interaction between the users and items, such as the ratings that a user gives to an item. Aggarwal et al. (1999) proposed a graph-theoretic algorithm in

which the similarity between two users is computed based on their shortest distance in the graph. The predicted rating that a target user may give to a target item is calculated based on the shortest direct paths between the target user and the others who have also rated the target item. Huang et al. (2004) applied an associative retrieval framework to explore the transitive associations among users through past transactions in order to estimate a target user's preference for a target item. Pucci et al. (2007) adaopted Google's PageRank Algorithm (Brin and Page, 1998) and proposed "Item-Rank," a random walk based scoring algorithm that can be used to rank items according to expected user preferences in order to recommend top-rank items. Baluja et al. (2008) also employed a random walk model on the video co-view graph to generate personalized video suggestions for users; this was once applied in YouTube's video suggestion engine. Graph-based approaches have the advantage of discovering new items, improving the novelty of recommendations, and addressing the problem of sparse ratings. However, it also requires extensive resources for setting up the graph representation and is computationally intensive.

2.2 Content-based Filtering

Content-based filtering is an outgrowth and continuation of information filtering research (Belkin and Croft, 1992). It recommends items similar to the target user's profile. A typical content-based filtering algorithm consists of three steps: *i*) content analyzer (or content representation), *ii*) user profile learning, and *iii*) content filtering (Mooney and Roy, 2000). Step *i*) models the features of items, whereas the Step *ii*) and Step *iii*) are usually connected with each other. The major difference between collaborative filtering and content-based filtering is that the former only uses the user-item ratings data to make predictions and recommendations, while the

latter relies on the features of users and items for predictions. For example, a music content-based recommender system will extract content features such as low-level timbre descriptors to determine item similarity, while many other domains (such as books, scholarly papers, movies, and apps) tend towards content features based on textual descriptions.

For textual items, the feature modeling has been widely studied for information retrieval where items are usually represented as a bag-of-words with "term frequency-inverse document frequency" (tf-idf) scores (Salton and McGill, 1986) or latent topic distribution (Steyvers and Griffiths, 2007). The latter has been proven to be more precise, notably latent Dirichlet allocation (LDA) (Blei et al., 2003) and its variants (Lin et al., 2013; Moshfeghi et al., 2011; Ramage et al., 2009; Wang and Blei, 2011).

Other notable content-based filtering works have been proposed over time. Pazzani and Billsus (1997) conducted a comprehensive experimental study comparing the performance of different classification techniques for content-based website recommendation. Billsus et al. (2000) developed a news recommendation agent that employs the simple k nearest neighbor classifiers (or "k-NN" for short), which is a lazy learner that finds the k nearest points from the training records to create a model of a target user's short term interest. Gutta et al. (2000) implemented a television show content-based recommendation approach using a Bayesian classifier. Christakou and Stafylopatis (2005) built a content-based movie recommender system by training three Neutral Networks for each user; each of which corresponded to "kinds," "stars," and "synopsis." Zhang and Koren (2007) improved the standard expectation-maximization (EM) algorithm to speed up the Bayesian learning process of content-based recommendation.

2.3 Social-based Recommendation

Another approach to computing similarity among items is through web-mining techniques, or exploiting information from social networks (*e.g.*, Facebook and Twitter), particularly follower/followee relationships. The basic idea is to, instead of using ratings-based similarity, utilize the sub-graphs of a user's social network (*i.e.*, the people that the target user is *following*) as "people prefer recommendations from people they know" (Bonhard and Sasse, 2006). Said et al. (2010) investigated a movie recommender system that has its own underlying social network and showed that the recommendation quality can be improved by utilizing user-to-user relationships. Another way of utilizing social networks is to view them as a "trust-based" network. Golbeck (2006) used a probabilistic matrix factorization framework that incorporates both the user-rating matrix and the users' trust in the social network to generate recommendations. Bedi et al. (2007) proposed a trust-based recommender system that uses the knowledge distributed over the network in the form of ontologies and employs the "web of trust" to generate recommendations. Ma et al. (2008) developed a factor analysis method based on the probabilistic graphical model that fuses the user-item matrix with the users' social trust networks by sharing a common latent low-dimensional user feature matrix. Jamali and Ester (2010) incorporated the mechanism of trust propagation into matrix factorization and showed that it leads to an increase in recommendation accuracy. Wang et al. (2010) proposed the use of the random walk model to capture the users' social influence similarity in order to predict users' opinions. Ma (2013) explored how to improve recommender systems using implicit social information, in which a general matrix factorization framework is employed to incorporate different implicit social information. Abisheva et al. (2014) combined user-centric data from Twitter with video-centric

data from YouTube to build a rich picture of *who* watches and shares *what* on YouTube, which could be used for video recommendation.

2.4 Hybrid Recommender Systems

Hybrid recommender systems are those that combine two or more recommendation techniques to minimize some of the issues that a single technique has and achieve some synergy between them. Combining different methods can be done using a number of ways:

1. **Weighted** — where the score of different recommendation components are combined numerically;
2. **Switching** — where the system chooses among recommendation components and applies the selected one;
3. **Mixed** — where recommendations from different recommenders are presented together;
4. **Feature Combination** — where features derived from different sources are combined and given to a single recommendation algorithm.

2.4.1 Weighted

The simplest design for a hybrid system is a weighted one. Each component of the hybrid system scores a given item and the system then combines the scores using a linear formula. Candidates are then sorted by the combined score and the top items are shown to the user. For example, the movie recommender system by Mobasher et al. (2003) made use of collaborative filtering and content-based filtering and combined the two components using a linear weighting scheme. Similarly, Claypool et al.

(1999) linearly combined collaborative filtering and content-based filtering in an online newspaper. This type of hybrid combines evidence from both recommenders in a static manner, and would therefore seem to be appropriate when the component recommenders have consistent relative power or accuracy across the product space. At the same time, its main limitation is that each component makes a fixed contribution to the score despite the possibility that recommenders will have different strengths in different parts of the product space. This suggests another hybrid in which the recommender systems switches between its components depending on the context.

2.4.2 Mixed

A mixed hybrid presents recommendations of its different components side-by-side in a combined list; there is no attempt to combine evidence between recommenders. The challenge in this type of hybrid recommender system is one of *presentation*: If lists are to be combined, how are rankings to be integrated? Typical techniques include merging based on predicted ratings or on recommender confidence. It is difficult to evaluate a mixed recommender using retrospective data. With other types of hybrids, we can use a user's actual ratings to decide if the right items are being ranked highly; but with a mixed strategy, especially one that presents results side-by-side, it is difficult to determine how the hybrid improves over its constituent components (Burke, 2007).

2.4.3 Switching

A switching hybrid is one that selects a single recommender from among its constituents based on the recommendation situation. For a different profile, a different recommender might be chosen. This approach takes into

account the problem that components may not have consistent performance for all types of users. However, it assumes that some reliable criterion is available on which to base the switching decision, in which the choice of this switching criterion is important. Researchers have used confidence values inherent in the recommendation components themselves as was the case with NewsDude (Billsus and Pazzani, 2000), while others have used external criteria (Nakagawa and Mobasher, 2003). The question of how to determine an appropriate confidence value for a recommendation is an area of research (Cheetham and Price, 2004). A switching recommender requires a reliable switching criteria — either a measure of the algorithm's individual confidence levels (that can be compared) or some alternative measure. The criterion must also be well-tuned to the strengths of the individual components (Burke, 2007).

2.4.4 Feature Combination

The idea of feature combination is to inject features of one source (such as collaborative recommendation) into an algorithm designed to process data with a different source (such a content-based recommendation). The features that would ordinarily be processed by an individual recommender are instead used as part of the input to the actual recommender. This is a way to expand the capabilities of a well-understood and well-tuned system, by adding new kinds of features into the mix (Basu et al., 1998; Mooney and Roy, 2000). The feature combination hybrid is not a hybrid in the sense that we have seen before (*i.e.*, that of combining components) because there is *only one* recommendation component. What makes it a hybrid is the knowledge sources involved; a feature combination hybrid borrows the recommendation logic from another technique rather than employing a separate component that implements it. For example, in the work of Basu

et al. (1998), the content-based recommender works in the typical way by building a learned model for each user, but user rating data is also combined with the product features. The system has only one recommendation component and it works in a content-based way, but the content draws from a knowledge source associated with collaborative recommendation.

The feature combination method has been used in many recent works, particularly in the winning solution of the Netflix Prize (Koren, 2009) as well as unified frameworks that crosses the domains of search and recommendation, such as the work by Wang et al. (2012). A common technique used by these works is Friedman (2001)'s Gradient Tree Boosting (GTB), an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems; it has also been used by the top performing algorithms in the *Learning To Rank Challenge*[2]. We will use this technique in Chapter 5 to combine the user and app features with the recommendation scores of the app recommendation techniques to produce a hybrid app recommender system.

2.5 Recommender Systems for Mobile Apps

Finally, we cover works on mobile app recommendation. In order to deal with the recent rise in the number of apps, works on mobile app recommendation are emerging. Some of these works focus on collecting additional information from the mobile device to improve recommendation accuracy. Xu et al. (2011) investigated the diverse usage behaviors of individual apps by using anonymized network data from a tier-1 cellular carrier in the United States. Yan and Chen (2011) and Costa-Montenegro et al. (2012) constructed app recommender systems by analyzing the usage patterns of users. Other works utilize external information to improve recommenda-

[2] `http://learningtorankchallenge.yahoo.com/workshop.php`

tion quality. Zheng et al. (2010) as well as Davidsson and Moritz (2011) made use of GPS sensor information to provide context-aware app recommendation. Lin et al. (2013) utilized the follower/followee information on Twitter to improve app recommendation in cold-start situations. Yin et al. (2013) considered behavioral factors that invoke a user to replace an old app with a new one, and introduced the notion of "actual value" (satisfactory value of the app after the user used it) and "tempting value" (the estimated satisfactory value that the app may have), thereby regarding app recommendation as a result of the contest between these two values. While the above works recommend apps that are similar to a user's interests, Bhandari et al. (2013) proposed a graph-based method for recommending serendipitous apps.

In addition, other ranking and prediction works that are remotely related to the domain of mobile apps include mining and predicting app usage behaviors (Liao et al., 2013) as well as ranking fraudulent apps in the app store (Zhu et al., 2013).

Chapter 3

Mobile App Recommendation Using Nascent Signals from Microblogs

This chapter looks at how we can use information from microblogs to improve on mobile app recommendation, particularly in the cold-start scenario.

3.1 Introduction

With the rise of social networking services such as Facebook[1] and Twitter[2], people broadcast and message friends, colleagues, and the general public about many different matters in a short and informal manner (Recuero et al., 2011). They do it often as the overhead of broadcasting such short messages is low. For instance, Twitter experienced several record-breaking events in 2012, such as having one million accounts created daily and having

[1] https://www.facebook.com
[2] https://www.twitter.com

175 million tweets sent out per day[3]. Additionally, social networks often include rich information about its users (Wu et al., 2011), such as posted user-generated content, user logs, and user-to-user connections (*e.g.*, Alice *follows* Bob). People use social networking services to talk about many subjects — including apps. An interesting possibility thus arises: Can we merge information mined from the rich data in social networks to enhance the performance of app recommendation? More formally, can we address the cold-start weaknesses of the proprietary user models of recommender systems (*e.g.*, the *user profiles* in an app store) by using nascent signals about apps from social networks (*e.g.*, the *user profiles* in Twitter)?

Through our case studies on the correlation and lag between social networks and formal reviews in app stores, we verified that the answer to this question is indeed yes. How then, would one go about capturing and encoding data from social networks? Through our observation of app stores, we note that the descriptions of some apps contain references to their Twitter accounts; by having a Twitter account, an app's developer or its organization can interact with its users on Twitter and market themselves, such as announcing new apps or updates. For example, within the descriptions in its Google Play[4] and iTunes[5] app stores, the "Angry Birds Star Wars" app has a line that says "follow @angrybirds[6] on Twitter." We also observed that app mentions in social networks can precede formal user ratings in app stores. This is important as it asserts that an early signal for app ranking (and thus recommendation) can be present in social networks. For example, Figure 3.1 shows that the "Evernote" app that was released in May 2012 had no ratings in the iTunes App Store for two months. It was

[3] "100 Fascinating Social Media Statistics and Figures From 2012," Huffington Post, accessed on Jan 10, 2013, `http://www.huffingtonpost.com/brian-honigman/100-fascinating-social-me_b_2185281.html`.

[4] `https://play.google.com/store/apps/details?id=com.rovio.angrybirdsstarwars.ads.iap`

[5] `https://itunes.apple.com/en/app/angry-birds-star-wars/id557137623`

[6] `http://twitter.com/angrybirds`

Figure 3.1: For two months since its release on the iTunes App Store, the "Evernote" app did not have any ratings. However, its Twitter account already had active tweets and followers. This shows that despite the cold-start, there is still information out there about the app, particularly on social networking services like Twitter.

only until July 2012 that the first few ratings started coming in. However, even before May 2012, Evernote's Twitter account already had more than 120,000 followers and 1,300 tweets. We take advantage of this active yet indirect information that is present in Twitter and use it to alleviate the cold-start problem that besets newly-released apps.

We integrate these findings into a novel approach to app recommendation that leverages on information from social networks (in specific, Twitter) to drive recommender systems in cold-start situations. By extracting *follower* information from an app's Twitter account, we create a set of *pseudo-documents* that contains Twitter-follower information; these pseudo-documents are different from the standard documents written in natural language. We then utilize latent Dirichlet allocation (LDA) (Blei et al., 2003) to construct latent groups of "Twitter personalities" from the pseudo-documents. By harnessing information from the linked topology of Twitter accounts, we can infer a probability of how likely a target user will like a newly released app — even when it has no official ratings.

We conduct extensive experiments and compare the usage of Twitter-follower information with other types of features, such as app genres, app developers, and text from app descriptions. In order to show that our approach excels not because of the use of Twitter information alone, we compare our approach with a non-LDA technique that also employs the

same information from Twitter. Finally, we combine our Twitter-follower feature with other features gleaned from app metadata through the use of gradient tree boosting (Friedman, 2001) and compare our approach with other state-of-the-art techniques. We show that our approach significantly outperforms these techniques.

3.2 Related Work

As the lack of ratings (*i.e.*, the cold-start) hinders the use of collaborative filtering techniques (Schein et al., 2002), various alternatives have been employed to overcome the problem. For example, Zhou et al. (2011) experimented with eliciting new user preferences by using decision trees with collaborative filtering. Moshfeghi et al. (2011) proposed a method for combining content features such as semantic and emotion information with ratings information for the recommendation task. Liu et al. (2011) identified representative users whose linear combinations of tastes are able to approximate other users. Likewise, many other works attempt to overcome the cold-start by finding radical ways of using proprietary user models and additional content. Another effective approach to the cold-start is to use latent factor models (Koren and Bell, 2011) that map users and items into a dense and reduced latent space that captures their most salient features. These models provide better recommendations than traditional neighborhood methods (Herlocker et al., 1999) as they reduce the level of sparsity and improve scalability (Koren, 2008). Notable latent factor models include probabilistic latent semantic analysis (PLSA) (Hofmann, 2004), principal component analysis (PCA) (Kim and Yum, 2005), artificial neural networks such as the restricted Boltzmann machine (RBM) (Salakhutdinov et al., 2007), and singular vector decomposition (SVD) (Takács et al., 2008). However, latent factor models have two major disadvantages in

recommendation tasks. Firstly, the learned latent space is not easy to interpret. Secondly, many latent factor models rely on other user ratings, which may be lacking if the dataset is sparse (Wang and Blei, 2011).

Our work differs from the ones mentioned above in that instead of using proprietary user models and content, we use information from Twitter (*i.e.*, non-proprietary content from social networks) and construct latent groups of "Twitter personalities" to predict recommendations under the cold-start. Although textual features have generally been a popular source of alternative data to substitute for the lack of ratings — such that even state-of-the-art techniques are primarily dependent on — it is not a universal solution as not all domains contain reliable textual data. Additionally, it is more realistic to rely on external social networks. Therefore, our work is unique in that it uses the "who *follows* whom" information on Twitter as its primary source of data, as textual features in the app domain are inherently noisy and unreliable (in contrast to the cases of movies or scholarly papers).

3.3 Our Approach

We first explain the kind of problem we address. Then we describe the relation between apps and Twitter-followers, and how we use them in our work. Next, we construct *pseudo-documents* and *pseudo-words* using data from users, apps, users' preferences, and Twitter-followers. Thereafter, we generate *latent groups* from the pseudo-documents, whereby a latent group represents the combined "interests" of various Twitter-followers. Finally, the set of latent groups is used as a crucial component of our algorithm to estimate the probability of a target user liking an app.

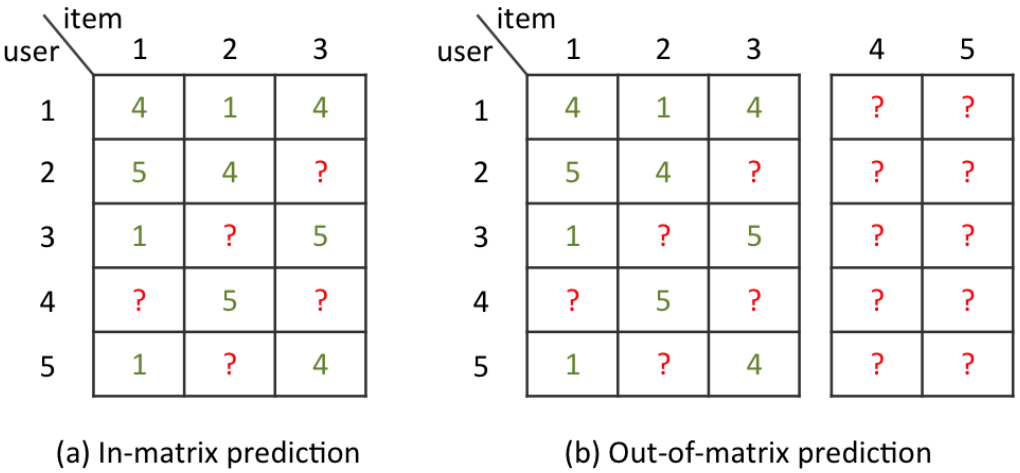


Figure 3.2: Difference between (a) in-matrix prediction and (b) out-of-matrix prediction.

3.3.1 Targeting the Cold-Start Problem

There are two types of cold-start problems in collaborative filtering: (a) in-matrix prediction and (b) out-of-matrix prediction (Wang and Blei, 2011). Figure 3.2(a) illustrates in-matrix prediction, which refers to the problem of recommending items that have been rated by at least one user in the system. This is the task that collaborative filtering researchers have often addressed (Aggarwal et al., 1999; Bell et al., 2007; Billsus and Pazzani, 1998; Breese et al., 1998; Hofmann and Puzicha, 1999; Koren and Bell, 2011; Salakhutdinov et al., 2007; Sarwar et al., 2001; Xue et al., 2005). On the other hand, Figure 3.2(b) illustrates out-of-matrix prediction, where newly released items (*e.g.*, items 4 and 5) have never been rated. Traditional collaborative filtering algorithms cannot predict ratings of items in the out-of-matrix prediction as they rely on user ratings, which are unavailable in this scenario. Our work focuses on this second, more challenging problem. Hereafter, we use "cold-start problem" to refer to "out-of-matrix prediction," for ease of reference.

3.3.2 Apps and their Twitter-Followers

Apps have textual descriptions displayed in their app store entries; some of these descriptions further contain links to the apps' official Twitter account (*i.e.*, *Twitter handle*). For example, the "Angry Birds" franchise contains a link to its Twitter handle (@angrybirds). From the handle, we can identify the IDs of Twitter-followers who follow the app. We note that by following an app's Twitter handle, the Twitter-followers subscribe to the tweets that are related to the particular app, which can be seen as an indicator of interest (Cha et al., 2010). Figure 3.3 illustrates the relation between users, apps, and Twitter-followers. By using information from the apps' Twitter-followers, we can construct "latent personalities" from two sources of data: the app store and Twitter. Using these latent personalities, our algorithm is able to recommend newly released apps in the absence of ratings[7] as shown in Figure 3.2(b).

Given that an app has a set of Twitter-followers, our approach estimates the probability that user u — defined by his or her past ratings — likes app a that is *followed* by Twitter-follower t (*i.e.*, Twitter-follower t follows app a's Twitter account):

$$p(+|t, u), \tag{3.1}$$

where "+" denotes the binary event that a user likes an app. Furthermore, as an app is represented by a set of its Twitter-followers, it is necessary to aggregate the probability in Equation (3.1) over the various Twitter-followers of app a. This allows us to estimate the probability of how likely the target user will like the app:

$$p(+|a, u). \tag{3.2}$$

[7]Although other types of information can be extracted from Twitter, such as the textual tweets and hashtags, we only focus on the Twitter-followers in this work as early experiments have shown that other types of information are noisy and potentially ambiguous.

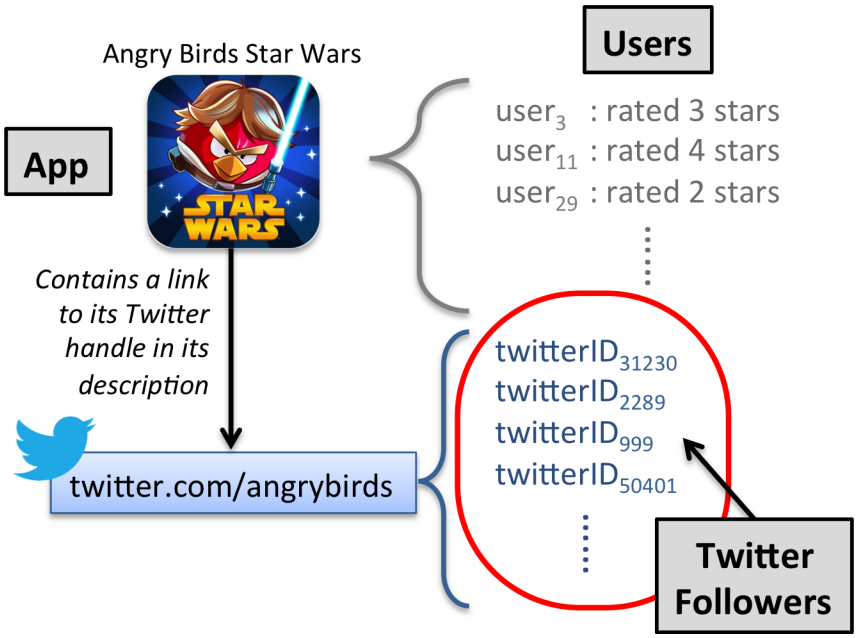


Figure 3.3: Instead of relying solely on the ratings of users, our approach also makes use of the Twitter IDs that *follow* the apps (red oval).

3.3.3 Pseudo-Documents and Pseudo-Words

In order to estimate the probability $p(+|a, u)$ in Equation (3.2), we build upon latent Dirichlet allocation (LDA) — a generative probabilistic model for discovering latent semantics that has been mainly used on textual corpora. Given a set of textual documents, LDA generates a probability distribution over latent "topics" for each document in the corpus, where each topic is a distribution over words. Documents that have similar topics share the same latent topic distribution. We adapt LDA for the purpose of collaborative filtering. As mentioned in Section 3.3.2, users download apps and apps may have Twitter-followers. Hence, user u and the Twitter-followers (of the apps that user u has downloaded) are analogous to a document and the words in the document, respectively. For the sake of clarity, we will call them *pseudo-documents* from now on as our "documents" actually do not contain natural language words in our work[8].

[8]The concept of *pseudo-documents* is similar to the idea mentioned in the original LDA paper by Blei et al. (2003) in which the authors state that "it is important to

Drawing on the parallelism, we formally define the following terms:

- We first assume that user u likes app a, and app a has a set of Twitter-followers $\{t_1, \ldots, t_n\}$ following its Twitter handle.
- A *pseudo-document* represents user u. Because of this, we use u to represent both pseudo-document and user.
- A *pseudo-word* is a "word" in pseudo-document u that corresponds to the ID of a Twitter-follower t. If user u has liked the apps a_1, a_2, and a_3, the pseudo-document u will then contain all the IDs of the Twitter-followers that are following the Twitter handles of apps a_1, a_2, and a_3. Note that there may be duplicated *pseudo-words* as a Twitter-follower t may be following more than one app's Twitter handle.

However, the problem of only considering the "liked" apps is in that LDA will indirectly assign higher probabilities to apps that many users liked. In other words, LDA will indirectly give high probabilities to popular apps. For example, suppose that two apps are judged the same number of times. The probability given by LDA will rank the two apps in order of their probability to be "liked," which we desire. In contrast, if the first app has been judged by all the users and half of them liked the app, it will have the same probability as another app that was judged by only half of the users but liked all the time, which is undesirable.

To address this popularity problem, we incorporate the magnitude of negative information as it indirectly allows us to account for the frequency of the whole judging group (*i.e.*, "liked" apps + "disliked" apps = total apps). In addition, this solution allows user groups to not only reflect the

note, however, that the LDA model is not necessarily tied to text, and has applications to other problems involving collections of data, including data from domains such as collaborative filtering, content-based image retrieval and bioinformatics."

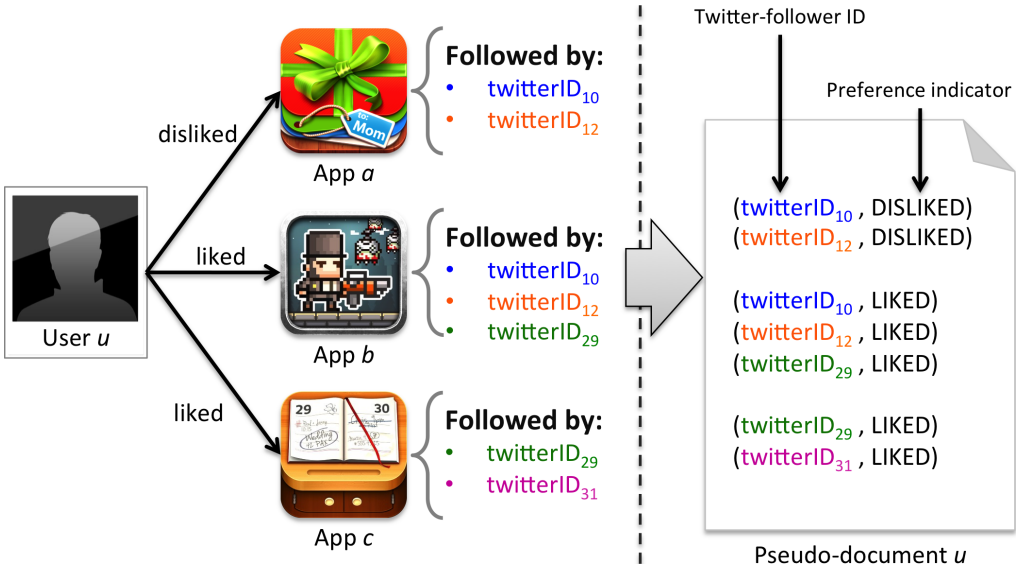


Figure 3.4: A *pseudo-document* is constructed based on information from a user, apps, Twitter-followers, and binary ("liked" or "disliked") preference indicators. A pseudo-document contains *pseudo-words*; each pseudo-word is represented as a *tuple* containing a Twitter-follower ID and a binary preference indicator.

Twitter-followers that appear in the apps that they like, but also in the apps that they dislike, thus providing richer information. We now formally define a pseudo-word:

- A *pseudo-word* is a "word" in pseudo-document u that contains the ID of a Twitter-follower t and its associated binary ("liked" or "disliked") preference indicator.

Figure 3.4 illustrates how a *pseudo-document* is constructed based on *pseudo-words*, which in turn are constructed based on the IDs of Twitter-followers and the binary preference indicators from users. Furthermore, in order to obtain the binary preferences, it is mandatory to convert the user ratings (*i.e.*, the 5-point Likert scale) into binary like/dislike indicators (see Section 3.4.2).

Note that the concept of pseudo-documents and pseudo-words does not apply exclusively to Twitter-followers; it can also be applied to other sources of information such as the app genres, app developers, and the

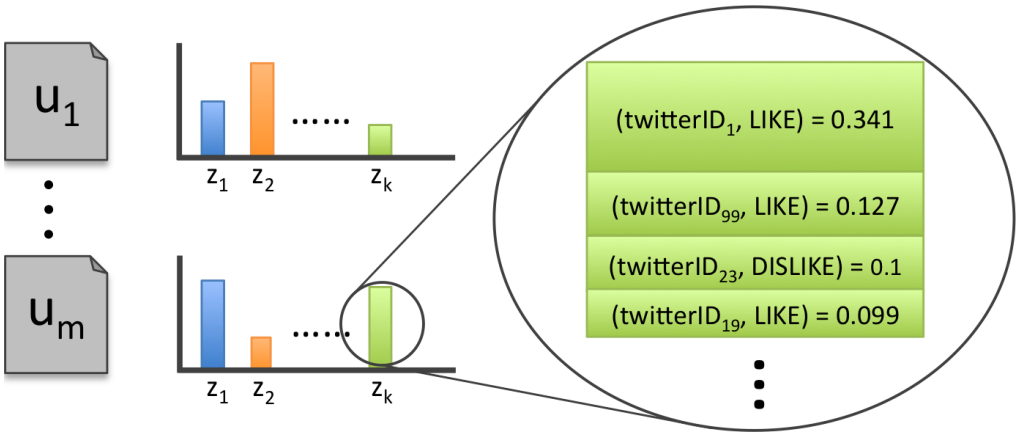


Figure 3.5: Given the set of pseudo-documents $\{u_1, \ldots, u_m\}$, LDA generates a probability distribution over *latent groups* for each pseudo-document, where each latent group z is represented as a distribution over pseudo-words. A pseudo-word is represented as a tuple containing a Twitter-follower ID and a binary preference indicator.

words in the app descriptions. For example, instead of using the IDs of Twitter-followers, we can also construct pseudo-words based on the IDs of app developers. We focus on Twitter-followers as our goal is to assess the effectiveness of this new source of data. In Section 3.5.1, we will create different sets of pseudo-documents based on different features (*i.e.*, Twitter-followers, app genres, app developers, and words) and identify the most effective feature in the recommendation of apps.

3.3.4 Constructing Latent Groups

Given the set of pseudo-documents $\{u_1, \ldots, u_m\}$, LDA can generate a probability distribution over *latent groups* for each pseudo-document, where each latent group is represented as a distribution over Twitter-follower IDs and binary preference indicators. Figure 3.5 illustrates this framework. By using the information on "which apps are followed by which Twitter-followers," we can estimate the probability of target user u liking app a by taking into account the IDs of the Twitter-followers following app a, and marginalizing over the different latent groups of pseudo-document u.

Given the set of tuples of pseudo-words $T_u = \{(t_1, d_1), \ldots, (t_n, d_n)\}$, where t_i and d_i are a Twitter-follower ID and its associated binary ("liked" or "disliked") preference indicator, respectively, we define LDA as a generative process that creates a series of tuples:

$$p(T_u|\alpha, \beta) = \int p(\theta|\alpha) \left(\prod_{i=1}^{n} \sum_{z=1}^{K} p(z|\theta) p(t_i, d_i|\beta_z) \right) \mathrm{d}\theta,$$

where K is the number of latent groups, θ follows the Dirichlet distribution of hyper-parameters α, and the latent group z follows a multinomial distribution given by β_z. The model is fully specified by α and β_z for each possible latent group z. Those hyper-parameters are learned by maximizing the likelihood of the dataset.

The LDA model is used to compute the probability that the presence of a Twitter-follower t indicates that it is "liked" $(+)$ (or "disliked" $(-)$) by user u, given user u's past interaction T_u and the learned parameters α and β. Hence, we get the following equation:

$$\begin{aligned} p(\pm, t|u) &= p(\pm, t|T_u, \alpha, \beta) \\ &= \sum_{z \in Z} p(\pm, t|z) p(z|u), \end{aligned} \tag{3.3}$$

where Z is the set of latent groups, $p(\pm, t|z)$ is computed from the per-topic word distribution of LDA, and $p(z|u)$ is computed from the per-document topic distribution of LDA.

3.3.5 Estimation of the Probability of How Likely the Target User Will Like the App

Our approach is based on a simple "averaging" method where the probability of how likely the target user will like the app is the expectation of how the Twitter-followers like the app. Given a set of Twitter-followers T,

the probability that user u likes app a is defined as follows:

$$\begin{aligned} p(+|a,u) &= \sum_{t \in T(a)} p(+,t|a,u) \\ &= \sum_{t \in T(a)} p(+|t,u)P(t|a), \end{aligned} \tag{3.4}$$

where $T(a)$ is the set of possible Twitter-followers following app a, in which we assume that: (i) Twitter-followers are examined one at a time to make a decision about whether an app is liked or disliked. In this case, t and t' are disjoint events whenever $t \neq t'$; (ii) when the Twitter-follower is known, the judgement does not depend on the app any more, *i.e.*, $p(+|t,a,u) = p(+|t,u)$; (iii) the fact that given a user and an app, there is no judgement involved, *i.e.*, $p(u,a) = p(u)p(a)$; (iv) the fact that an app has a given Twitter-follower is independent from the user, *i.e.*, $p(t|a,u) = p(t|a)$.

Equation (3.4) is then reduced to the estimation of two quantities:

1. the probability that user u likes app a given that app a has Twitter-follower t, *i.e.*, $p(+|t,u)$, and

2. the probability of considering Twitter-follower t given app a, *i.e.*, $p(t|a)$.

$p(+|t,u)$ is straightforward to estimate as it can be rewritten as:

$$p(+|t,u) = \frac{p(+,t|u)}{p(+,t|u) + p(-,t|u)},$$

where $p(+,t|u)$ and $p(-,t|u)$ are derived from LDA in Equation (3.3), which is the probability that Twitter-follower t occurs in an app that is liked (or disliked) by user u.

As for how to compute the probability $p(t|a)$, we explore the following two ways:

(I) Assume a uniform distribution over the various Twitter-followers present in app a

This framework is defined as follows:

$$p(t|a) = \begin{cases} \frac{1}{\#T(a)} & \text{if } t \text{ is a Twitter-follower of app } a \\ 0 & \text{otherwise,} \end{cases} \tag{3.5}$$

where $T(a)$ and $\#T(a)$ denote the set of Twitter-followers following app a and the set cardinality, respectively.

(II) Give more importance to influential Twitter-followers

In this alternative framework, we compute the influence of Twitter-followers by using TwitterRank (Weng et al., 2010). Let $TR(t)$ be the TwitterRank score of Twitter-follower t. With that, for each Twitter-follower t, we retrieve its TwitterRank score and normalize it with the scores obtained by the other Twitter-followers following app a. That is:

$$p(t|a) = \begin{cases} \frac{TR(t)}{\sum_{t' \in T(a)} TR(t')} & \text{if } t \text{ is a Twitter-follower of app } a \\ 0 & \text{otherwise,} \end{cases} \tag{3.6}$$

where $T(a)$ denotes the set of Twitter-followers following app a.

Prior tests on our dataset have shown that there is no significant difference in performance between (I) and (II) above. This is because the Twitter-followers that we get (note that each latent group consists of Twitter-followers) are generally not prominent, influential people. Rather, they are the average users on Twitter. Therefore, even if we use TwitterRank, the influence of each Twitter-follower eventually converges into the uniform distribution. We thus adopt the uniform distribution defined by Equation (3.5) in our evaluation for its simplicity.

3.4 Evaluation Preliminaries

We preface our evaluation proper by detailing how we constructed our dataset, our settings for the dataset and the LDA model, and how we chose our evaluation metric.

3.4.1 Dataset

We constructed our dataset by crawling from Apple's iTunes App Store and Twitter during September to December 2012. The dataset consists of the following three elements:

1. **App Metadata**. These include an app's ID, title, description, genre, and its developer ID. The metadata is collected by first getting all the app IDs from the iTunes App Store, and then retrieving the metadata for each app via the iTunes Search API. The metadata is the source for the genre, developer, and word features mentioned at the end of Section 3.3.3.

2. **Ratings**. For each app, we built a separate crawler to retrieve its reviews from the iTunes App Store. A review contains the app's ID, its rating, the reviewer's ID, the subject, and the review comments. This is the source of the rating feature.

3. **Related Twitter IDs**. We used two methods to collect app-related Twitter IDs. The first way is to get the IDs that follow an app's Twitter account. We scanned through each app's description to identify its Twitter handle. For each app's Twitter handle, we used Twitter's API to search for every Twitter ID following its handle. The second method uses Twitter's Streaming API to receive tweets in real-time. To retrieve tweets that are related to apps, we only kept tweets that contain hyperlinks to apps in the iTunes App Store, and stored the

Twitter IDs (who wrote the tweet) as well as the app IDs that were mentioned in the tweets.

Altogether, we collected 1,289,668 ratings for 7,116 apps (that have accounts on Twitter) from 10,133 users. The user-item ratings matrix has a sparsity of 98.2%. We also collected 5,681,197 unique Twitter IDs following the apps in our dataset. With respect to app ratings and the number of related Twitter-followers per app, we restrict that each user gives at least 10 ratings and each Twitter ID is related to at least 5 apps, respectively. On average, each user has rated 26 apps — ranging from a minimum of 10 rated apps to a maximum of 271.

3.4.2 Experimental Settings

In order to train the LDA model, we require binary relevance judgements to convert the user ratings to binary preference indicators, as well as two sets of hyper-parameters — namely, the number of latent groups K and the priors α and β.

We performed per-user normalization of the 5-point Likert scale ratings when converting them to the binary like/dislike values required by our LDA application. This is because the average rating for different users can vary significantly (Koren, 2008). Let $r_{avg(u)}$ and $r_{u,a}$ be user u's average rating among all apps and user u's rating for app a, respectively. The normalized rating is thus $r_{n(u,a)} = r_{u,a} - r_{avg(u)}$, where if $r_{n(u,a)} \geq 0$, the rating is converted to a "like" $(+)$, or "dislike" $(-)$, otherwise.

We set the priors α and β as proposed in (Misra et al., 2008). The number of latent groups has a crucial influence on the performance of the LDA approach. We used the likelihood over a held out set of training data to find the relevant number of latent groups. We tried several settings for K (*i.e.*, the number of latent groups), namely 10, 20, 35, 50, 80, 100, 120, 150,

and 200. The final number of latent groups was selected by maximizing the likelihood of observations over the development set.

In order to simulate the cold-start, we selected 10% of the apps which were then the held out set for all users (*i.e.*, we removed their ratings in the training set). Therefore, each user has the same set of within-fold apps and we can guarantee that none of these apps are in the training set of any user. We performed 10-fold cross validation where in each fold, we used 70% of the apps to train the LDA model, 20% to identify the number of latent groups for LDA (*i.e.*, the development set), and the remaining 10% as test data.

3.4.3 Evaluation Metric

Our system outputs M apps for each test user, which are sorted by their probability of liking. We evaluate the algorithms based on which of these apps were actually downloaded and liked (*i.e.*, the normalized rating of the test user). This methodology leads to two possible evaluation metrics: precision and recall. However, a missing rating in training is ambiguous as it may either mean that the user is not interested in the app (negative), or that the user does not know about the app (truly missing). This makes it difficult to accurately compute precision (Wang and Blei, 2011). But since the known ratings are true positives, we feel that recall is a more pertinent measure as it only considers the positively rated apps within the top M, *i.e.*, a high recall with a lower M will be a better system. We thus chose Recall@M as our primary evaluation metric. Let n_u and N_u represent the number of apps the user likes in the top M and the total number of apps the user likes, respectively. Recall@M is then defined as their ratio: n_u/N_u. We compare systems using average recall, where the average is computed over all test users.

3.5 Experiments

As our approach specifically attempts to address the cold-start, we work on the specific scenario when a new set of apps is released and users have yet to rate them (shown in Figure 3.2(b)). The goal of our experiments is to answer the following research questions:

RQ1 How does the performance of the Twitter-followers feature compare with other features, such as the app genres, app developers, and words in the app descriptions? Can we obtain better recommendation accuracy by combining them?

RQ2 How does our proposed method compare with other state-of-the-art techniques?

RQ3 Do the latent groups make any sense? If so, what can we learn from them?

3.5.1 Comparison of Features (RQ1)

We benchmark how the signal from Twitter-followers affects performance in comparison to other sources of data. We compare the Twitter-followers feature (hereafter, T) with the other features: app genres (G), app developers (D), and words in the app descriptions (W). To perform this comparison in a fair manner, for each of these four features, we constructed a set of pseudo-documents that contains a set of *feature-related* pseudo-words (see Section 3.3.3).

Additionally, we assessed the effectiveness of these features in combination; we combine multiple features (*i.e.*, Twitter-followers, genres, developers, and words) through the use of gradient tree boosting (GTB) (Friedman, 2001). We also performed ablation testing where we removed features from the combined feature set to determine the importance of each feature. The

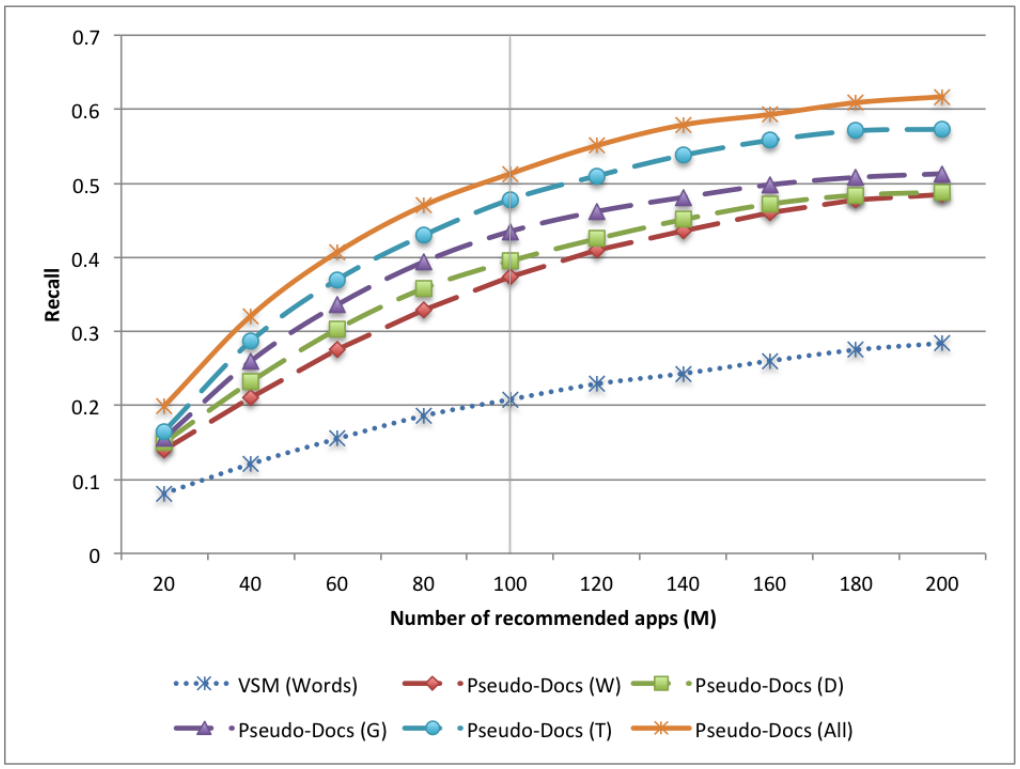


Figure 3.6: Recall obtained by different individual features (dashed lines), as well as our method that combines all features (solid line). The baseline vector space model (VSM), using the app description word vocabulary is also shown (dotted line). The vertical line marks model performance at $M = 100$ (*cf.* Table 3.1).

features given to GTB are a set of probabilities defined by Equation (3.4). In GTB, we set the maximum number of trees and maximum tree depth to 2000 and 3, respectively, and used the least-squares regression as a cost function.

Results. Figure 3.6 shows the results of the first experiment, which compares the overall performance between features (words (W), developers (D), genres (G), Twitter-followers (T), and all features (All) combined) when we vary the number of returned apps $M = 20, \ldots, 200$. Fixing $M = 100$, we ablated individual features from our combined method and show the results in Table 3.1. The results are quite consistent. In the overall comparison over all ranges of M, the individual feature of Twitter-followers gives the best individual performance, followed by genres, developers, and

Table 3.1: Recall levels in our feature ablation study at $M = 100$. TGDW and individual feature (T, G, D, W) performances in Figure 3.6 are also shown.

Feature	**R@100**
All features (TGDW)	**0.513**
All, excluding Twitter-followers (GDW)	0.452
All, excluding Genres (TDW)	0.491
All, excluding Developers (TGW)	0.498
All, excluding Words (TGD)	0.507
Twitter-followers (T)	0.478
Genres (G)	0.435
Developers (D)	0.395
Words (W)	0.373

words in the app descriptions. From the combination and ablation study in Table 3.1, we see that all features are necessary for the optimal results. Matching the results in Figure 3.6, Table 3.1 also shows that the removal of the best (worst) individual features leads to the corresponding largest (smallest) drop in recall.

It may be surprising that the developer (D) and genre (G) features are more effective than words in the app description (W). We observe that users may favor developer brands; possible causes could be that the user recognizes the brand, or that the apps themselves may promote sister apps that are made by the same developer. In addition, some apps complement one another. For example, "Google Chrome," "Gmail," and "Google Maps" form a complementary set. Also, the genres of apps may correlate with download behavior. Figure 3.7 shows the distribution of app genres, and indeed we observe that the "Games" genre dominates the distribution. This indicates that users often download apps that belong to the "Games" genre.

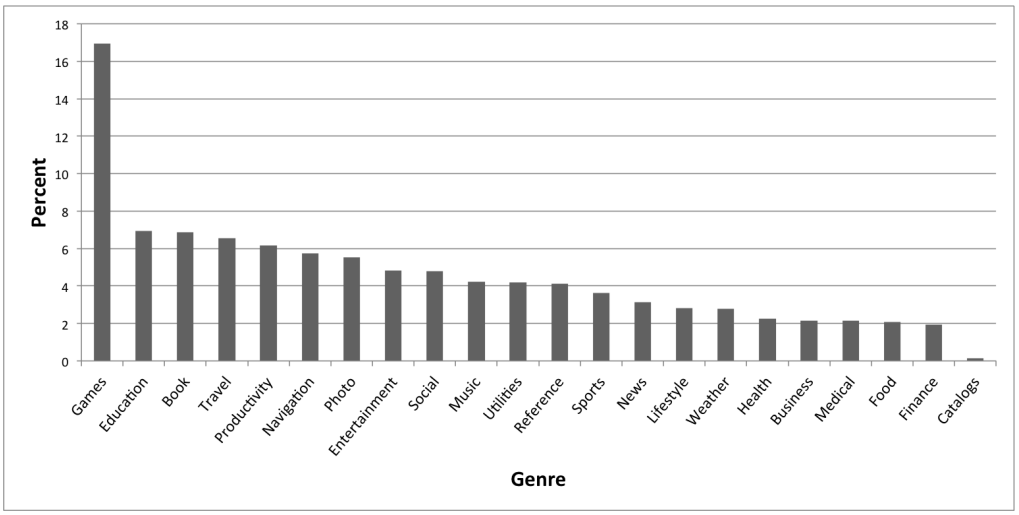


Figure 3.7: Distribution of app genres within our dataset.

From a practical standpoint, the most straightforward way to recommend apps in a cold-start would be to use the textual descriptions in a content-based filtering system, as in our (W) system. But it performs the worst among the four individual features. Why do words in app descriptions (W) perform the least well?

Carrying out a more detailed inspection, we find that app descriptions do not give informative hints about the app's role; rather, they are more focused on self-promotion as many apps boast about the reviews that they have received in their app descriptions. Figure 3.8 is a screenshot of an app description that demonstrates this fact. In addition, according to Ayalew (2011), users pay more attention to screenshots instead of descriptions, which suggests that the information from app descriptions is not as useful. This result also further explains why text-dependent baselines such as CTR and LDA did not perform much better (*cf.* the next section on RQ2).

Features aside, we also assessed whether our use of the LDA-based pseudo-document method is an important factor in recall performance. We compared a straightforward use of the same data with a standard vector space model (VSM) where we used the same app description words (W)

Figure 3.8: A screenshot of an app description that illustrates why word features may not be effective as it largely boasts about endorsements received.

to build a vector of standard tf-idf weighted words to represent each app. Figure 3.6 also shows this result in the bottom two lines. We see that the pseudo-document use of words (W) greatly outperforms the VSM-based version (VSM). A two-tailed t-test at $M = 100$ shows that the improvement is statistically significant ($p < 0.01$). This validates our LDA-based pseudo-document approach.

3.5.2 Comparison Against Baselines (RQ2)

As we focus on the cold-start problem, we did not consider other well-known recommender techniques that require user ratings, such as matrix factorization or latent factor models. We compare our approach with four baselines. The first two baselines are VSM-based — the first is the app description-based VSM recapped from RQ1 (*i.e.*, "VSM (Words)") while the second uses the IDs of Twitter-followers as its vocabulary (*i.e.*, "VSM (Twitter)"). The VSM (Twitter) baseline evaluates whether our LDA-based approach of using the Twitter-follower data betters the simpler VSM

method. The third baseline is the LDA model, which is equivalent to using our pseudo-documents model on words in app descriptions. Lastly, as a much stronger baseline, we show the performance of a re-implementation of Wang and Blei's (Wang and Blei, 2011) collaborative topic regression (CTR) model — a state-of-the-art collaborative filtering algorithm that can also make predictions in cold-start scenarios. Furthermore, in order to study the impact of sparsity on our models, in a separate experimental condition, we randomly removed some ratings from the training set so that the maximum number of rated apps per user was 15, which represents a sparser environment.

Results. Figures 3.9 and 3.10 plot the results of this set of experiments when we vary the number of returned apps $M = 20, \ldots, 200$. We observe that our approach that combines various pseudo-documents using GTB consistently and significantly outperforms other models (at $p < 0.01$ on the full dataset), particularly the CTR model that is the best among all baselines. This shows that we can achieve significant improvement in recommendation accuracy by integrating multiple sources of information. We also observe that our model of using pseudo-documents with Twitter data alone (*i.e.*, "Pseudo-Docs (Twitter)" in Figures 3.9 and 3.10) outperforms other models. This validates our observation that Twitter is indeed a good source of information to address the cold-start in app recommendation. It also indirectly points out that the textual features are less effective in the app domain, which is obvious from the performances of the CTR and LDA models that solely rely on textual data. We also note that the performance of CTR is fairly similar to LDA. This is due to the fact that the matrix-factorization component of CTR cannot perform recommendation in the cold-start. Therefore, like LDA, its recommendation is based entirely on content.

We also observe that between the two models that only use Twitter

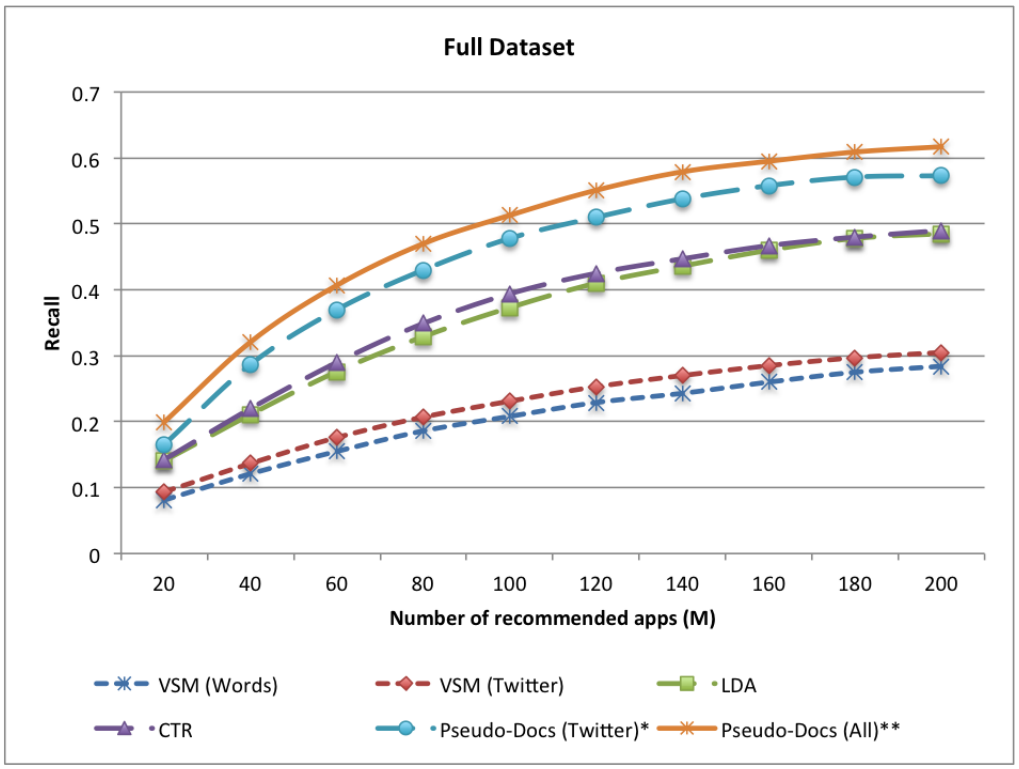


Figure 3.9: Recall varying the number of recommendations on the **full** dataset. "*" and "**" denote statistically significant improvements over the best baseline (CTR) at $p < 0.05$ and $p < 0.01$, respectively.

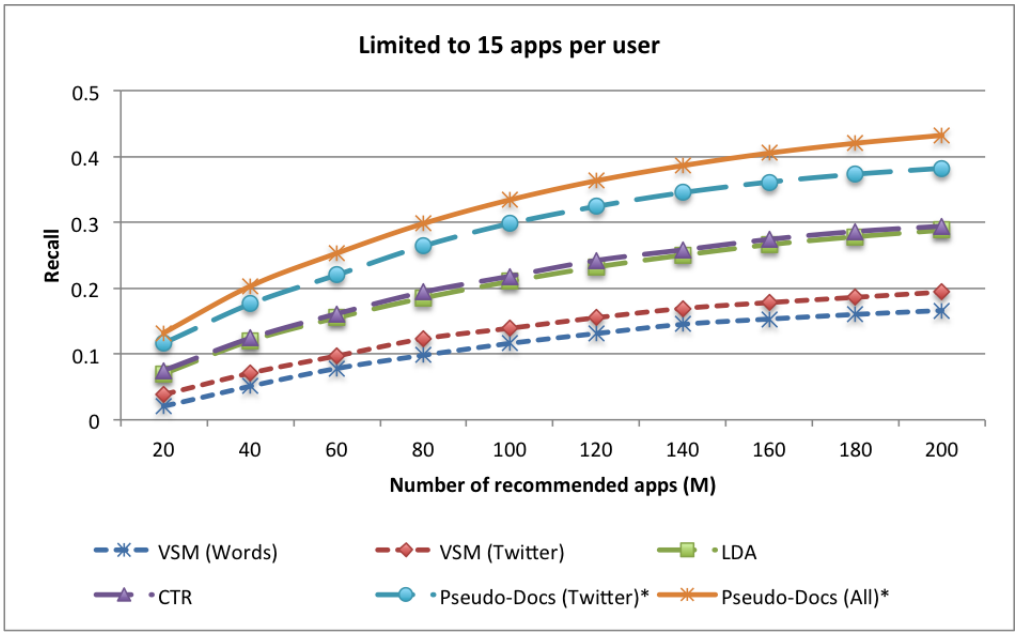


Figure 3.10: Recall varying the number of recommendations on the **sparse** dataset. "*" and "**" denote statistically significant improvements over the best baseline (CTR) at $p < 0.05$ and $p < 0.01$, respectively.

features (*i.e.*, "VSM (Twitter)" and "Pseudo-Docs (Twitter)"), our model significantly outperforms the VSM model. This again validates our earlier findings that our method's performance is not just due to the use of new data, but also how we make use of it. Another important observation is that under a sparser cold-start environment where we limit our training data to 15 apps per user at maximum, there is an overall drop in recall, which is expected. However, we note that our model (both using Twitter feature alone and multi-features) still outperforms all of the baselines. This indicates that using Twitter features is more robust than using app descriptions under sparse conditions, as the IDs of the Twitter-followers are independent from the apps — misleading or absent app descriptions do not directly influence the Twitter-followers.

3.5.3 Analysis of Latent Groups (RQ3)

We use LDA and the notion of Twitter users as pseudo-words to achieve good recall. A natural question to ask is whether the latent groups discovered by LDA (from the individual features of Twitter-followers, genres, and developers) have any meaning. We observe interesting points for the developer (D) and Twitter-follower (T) features.

For the developer feature, in most latent groups, the developers who have received a substantial number of ratings for all their apps are different from the independent ("indie") developers. This is due to the fact that these developers usually have apps that users "liked," whereas the indie developers sometimes have apps that users "disliked" as well as apps that users "liked." This coincides with our hypothesis about brand loyalty, as the developers with a substantial number of ratings either have created a large number of apps (such as EA Games) or have created a small number of well-received apps (such as the apps by Google). We also observe that the

Latent Group 1	Latent Group 2	Latent Group 3
Top genres: • Books (45%) • Education (33%) • Games (13%) **Example apps:** • The Cat in the Hat (Books) • Christmas Cutie (Books) • Happy Earth Day, Dear Planet (Books) • Friendly Shapes (Education) • There's No Place Like Space! (Education) • Pasta Crazy Chef (Games) • Gingerbread Dress (Games)	**Top genres:** • Music (92%) **Example apps:** • BeatStudio (Music) • AmpKit+ (Music) • GuitarStudio (Music) • Everyday Looper (Music) • Mixr DJ (Music) • KORG iELECTRIBE (Music) • KORG iPolysix (Music) • Pro Metronome (Music) • Chord Detector (Music)	**Top genres:** • Games (77%) • Photo & Video (12%) **Example apps:** • Another World 20th Anniversary (Games) • Paper Monsters (Games) • Stickman Cliff Diving (Games) • Lili (Games) • Snoopy's Street Fair (Games) • Gizmonauts (Games) • InstaBooth+ (Photo & Video) • ArtStudio for iPad (Photo & Video)
Nosy Crow Apps (twitter.com/nosycrowapps) Nosy Crow creates children's books and apps. You may know our 3-D Fairytale apps, The Three Little Pigs & Cinderella.	**Derek Jones (twitter.com/MusicInclusive)** Indie music publishing label, studio & brand. Blues&Rock, Progressive&Funk, Jazz&Fusion, Alternate&Christian, Classical, Education & a lot in-between too!	**Sarah Thomson (twitter.com/SarahLuvsVGames)** Video games warrior, lover of life, eternal student of the universe, drinker of Kombucha, Baroness of PlayStation Mobile.
The iMums (twitter.com/TheiMums) Four mums dedicated to reviewing apps and technology products for children to help educate their parents about the variety available. Loads of giveaways too!	**Chip Boaz (twitter.com/iosmusicandyou)** I'm a musician based in the San Francisco Bay Area with an interest in using my iPad, iPhone, & iPod to make music. Follow my iOS adventures @ iOS Music And You	**JasonLeeNester (twitter.com/JasonNester)** I am a Multimedia developer working at Kent State University! I also do art services for the game industry as well as run a small indie game company, True Media.
Mums with Apps (twitter.com/momswithapps) Supporting family-friendly developers seeking to promote quality apps for kids and families.	**Dave Gibson (twitter.com/MicroTrackdB)** Creator of MicroTrack dB, a music making app for iOS and Samsung bada. Musician, writer, audio engineer and synth nerd. http://www.facebook.com/microtrackdb	**Agalag iOS Games (twitter.com/AgalagGames)** Agalag Games is an independent iOS game studio. Our aim is to create fun innovative and casual iPhone games which we really like and want to play. Publisher.
Charly James (twitter.com/CharlyJames2) Div. Mom of 2 w/varying SN & medical d/x. dandelion moms; A4 Free Apps @CharlyJames4; Ellie's Games; Fernandez Design.	**Ashley Elsdon (twitter.com/IamAshleyElsdon)** Everything from Mobile Music Creation, geekery, tech, art and Doctor Who! http://www.ashleyelsdon.com	**Samadhi Games (twitter.com/SamadhiGames)** Hi! Samadhi Games LLC is an Indie Developer of iOS, Android and Desktop Apps. Arizona · http://www.samadhigames.com
Next is Great (twitter.com/nextisgreat) We create and develop brain teasing educational iOS apps for kids and teenagers. Check out Pirate Trio Academy and Geek Kids.	**Andrew Wardell (twitter.com/andrewwardell)** Nostalgic futurist, amateur photographer, sax-playing synthesist, musical mountain-biking metacomic. More than just a bag of salty water...	**Finger Arts: App Dev (twitter.com/fingerartsgames)** We develop cool & innovative iPhone, iPod Touch & iPad Games. Rocking the charts in iTunes: Sudoku 2, Hangman RSS, 4 in a Row & now Solitaire :)

Figure 3.11: The top 3 latent groups; each group shows the top 5 Twitter-followers and their public profile.

clustered developers are classified into the same genre or strong competitors like Facebook and Twitter.

In order to understand why Twitter-followers work best, we scrutinized the latent groups discovered by LDA at the optimal performance point of $K = 120$. Each group consists of Twitter-followers; and each Twitter-follower follows at least 5 apps in our dataset. We then manually visited the Twitter pages of the top 5 Twitter-followers in each of the 120 latent groups, and then verified their profile descriptions and their latest tweets. We observe that more than 95% of these Twitter-followers exhibit consistent interest in apps. This shows that our approach accurately distinguishes between Twitter users who have implicit/explicit interest in apps and regular Twitter users. Our approach assigns low probabilities to Twitter users who have little or no correlation to apps, effectively filtering out their influence as noise. To illustrate the quality of the latent groups, we performed an additional level of micro-analysis. We selected the 3 most important latent groups among the 120 groups based on the expectation of the probability of each latent group over the set of training data (*i.e.*, the pseudo-documents), along with their corresponding top 5 Twitter individuals' public profile (Figure 3.11). The top-most row in Figure 3.11 shows the top genres and examples of apps in each of the top 3 latent groups. We see that the apps in each group coincides with the Twitter profiles of the top 5 individuals in the same group. Latent Group 1 is composed of family-oriented Twitter users who also download family-oriented apps. Such a latent group would be difficult to describe if we were to use genres alone, as this group consists of a non-discrete mix of children-friendly apps. In Latent Group 2, we see that this group consists of professional music-creation apps, which also coincides with the type of Twitter users who are either actual musicians or people interested in creating music. This is in contrast with the "music" genre in app stores, which is in some sense too

diverse as it may refer to music-player apps, music-streaming apps, or trivial music-making apps. Lastly, Latent Group 3 captures games that are of a light-hearted, indie nature. We can also relate the Twitter users to the downloaded apps, which are difficult to describe using genres or words alone. In short, our Twitter-follower feature is able to capture the personalities of Twitter-individuals. Therefore, even when an app does not have user ratings, our system can still provide relevant recommendation based on information about *who is following* the app's Twitter account.

3.6 Conclusion

By taking advantage of the unique property of apps and their corresponding Twitter profiles, we identify Twitter-followers of the Twitter profiles of apps. Pseudo-documents are then created to represent users, where each pseudo-document contains the IDs of Twitter-followers of the apps that a user has previously downloaded. Thereafter, LDA is applied to the set of pseudo-documents to generate latent groups which is then used in the recommendation process. By combining the feature of Twitter-followers with other features based on various app metadata such as genre, developer, and the words in the app description, we can generate a much more accurate estimation of how likely a target user will like an app. Experimental results show that features extracted from Twitter consistently and significantly outperform state-of-the-art baselines which rely on (potentially misleading) textual information distilled from app descriptions. This also shows that *follower* information from Twitter helps us discover valuable signals that is effective in alleviating the cold-start situation.

Chapter 4

Mobile App Recommendation Using Version Features

This chapter looks at how we can use version features in the mobile app domain as content features for the purpose of improving recommendation.

4.1 Introduction

Another unique characteristic about the app domain is that, unlike conventional items that are static, apps change and evolve with every revision. Thus, an app that was unfavorable in the past may become favorable after a version update. For example, Version 1.0 of App X did not interest a user at first, but a recent update to Version 2.0 — which promises to provide the functionality of high definition (HD) video capture — may arouse his interest in the revised app. A conventional recommender system that regards an app as a static item would fail to capture this important detail. This is why it is vital for app recommender systems to process nascent

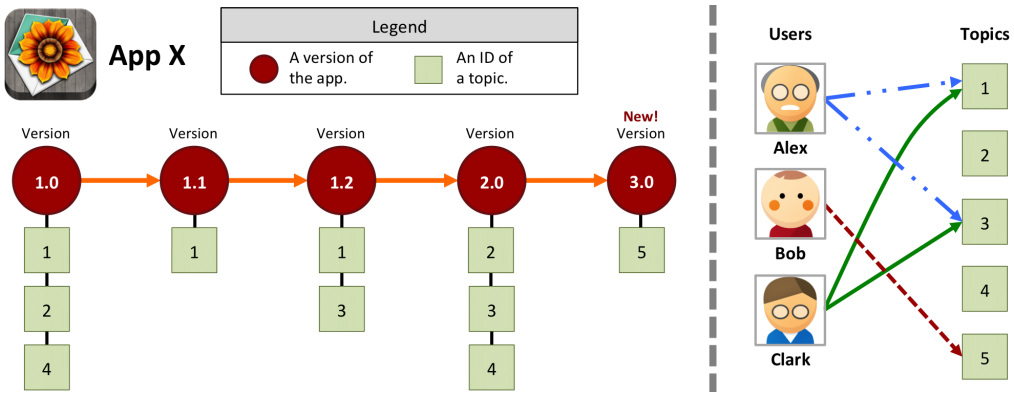


Figure 4.1: App X has five versions (red circles, on the left). The contents of each version is represented by a set of topics (green squares) in which each version consists of at least one topic. At the same time, based on the consumption history of users, we model them by identifying which topics they are interested in (on the right).

signals in version descriptions to identify desired functionalities that users are looking for.

We focus on the uniqueness of the app domain and propose a framework that leverages on *version features*; *i.e.*, textual descriptions of the changes in a version, as well as version metadata. First, with the help of semi-supervised topic models that utilize these features, we generate latent topics from version features. Next, we discriminate the topics based on genre information and use a customized popularity score to weight every unique genre-topic pair. We then construct a profile of each user based on the topics, and finally compute a personalized score of recommending the latest version of an app to a target user. Furthermore, we show how to integrate this framework with existing recommendation techniques that treat apps as static items.

Figure 4.1 provides an overview of our approach. App X has five different versions (1.0, 1.1, 1.2, 2.0, and 3.0). Each version is characterized by a set of latent topics that represents its contents, whereby a topic is associated with a functionality, such as the ability to capture HD videos.

For instance, Version 1.0 has Topics 1, 2, and 4; whereas Version 3.0 only has Topic 5. At the same time, based on a user's app consumption history, we can model which topics they are interested in. Therefore, if Bob has a keen interest in Topic 5, the chance that he adopts App X at Version 3.0 would be higher because Topic 5 attracts Bob's interest. Likewise, there is a higher chance of both Alex and Clark adopting App X at Version 1.2 because Topics 1 and 3 attract their interests.

We show that the incorporation of version features complements other standard recommendation techniques that treat apps as static items, and this significantly outperforms state-of-the-art baselines. Our experiments identify which topic model best utilizes the available version features to provide the best recommendation, and examine the correlation between various version metadata and recommendation accuracy. Furthermore, we provide an in-depth micro-analysis that investigates: *i*) whether our approach recommends relevant apps at the most suitable version; *ii*) what information can we gather by scrutinizing the latent topics; and *iii*) which is the most influential version-category. To the best of our knowledge, this is the first work that investigates version features in recommender systems. Our contributions are summarized as follows:

- We show that version features are important in app recommendation, as apps change with each version update, unlike conventional static items. This ultimately influences the needs of users and the recommended apps.

- We show how to synergistically combine our version-sensitive method with existing recommendation techniques.

4.2 Related Work

Works in information retrieval (IR) have also handled items that change and evolve. For example, past works have also viewed Web pages as entities that evolve over time. Keyaki et al. (2012) explored XML Web documents (*e.g.*, Wikipedia articles) that are frequently updated, and proposed a method for fast incremental updates for efficient querying of XML element retrieval. This is different from our work as they dealt with items only, whereas our method also generates *personalized* recommendation of items for users, *i.e.*, our work also considers the users. Furthermore, our work focuses on a secondary item unit — version updates — which is a separate entity from primary item (*i.e.*, the app). Liang et al. (2012) proposed a method to capture the temporal dynamics of relevant topics in micro-blogs (*e.g.*, Twitter) where a topic centers around a certain theme such as the U.S. presidential election or Kate Middleton's baby which, in the micro-blogging community, may change quickly with time. Our work differs from theirs as the "items" in their system are the topics, which is an indefinite discourse. Apps, on the other hand, are definite items that users download and use. Wang and Zhang (2013) explored the problem of recommending the right product at the right time, which uses a proposed opportunity model to explicitly incorporate time in an e-commerce recommender system. Their work explores the time of purchase, and does not focus on items that change with time.

In summary, our work differs from the previous works in that the nature and requirements of app recommendation — with respect to version information — differs from the retrieval of Web articles, topic recommendation in micro-blogs, and the time of recommendation.

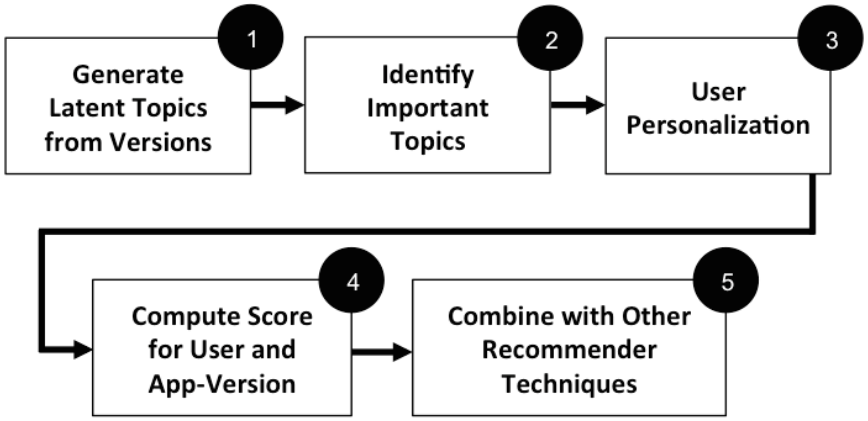


Figure 4.2: Overview of our framework.

4.3 Our Approach

Our framework processes the version texts and metadata to decide whether a particular version of an app entices a target user. As shown in Figure 4.2, we first generate latent topics from version features with semi-supervised topic models in order to characterize each version. Next, we discriminate the topics based on genre metadata and identify important topics based on a customized popularity score. Following that, we incorporate user personalization, and then compute a personalized score for a target user with respect to an app and its version. Our system then recommends the top k target apps:

$$A : a \in \arg\max_k(score(d(a, v), u)), \tag{4.1}$$

where an app a and its specific version v are treated as a tuple that characterizes a document d, and is scored with respect to a target user u. Lastly, we explain how to integrate this framework with existing recommendation techniques.

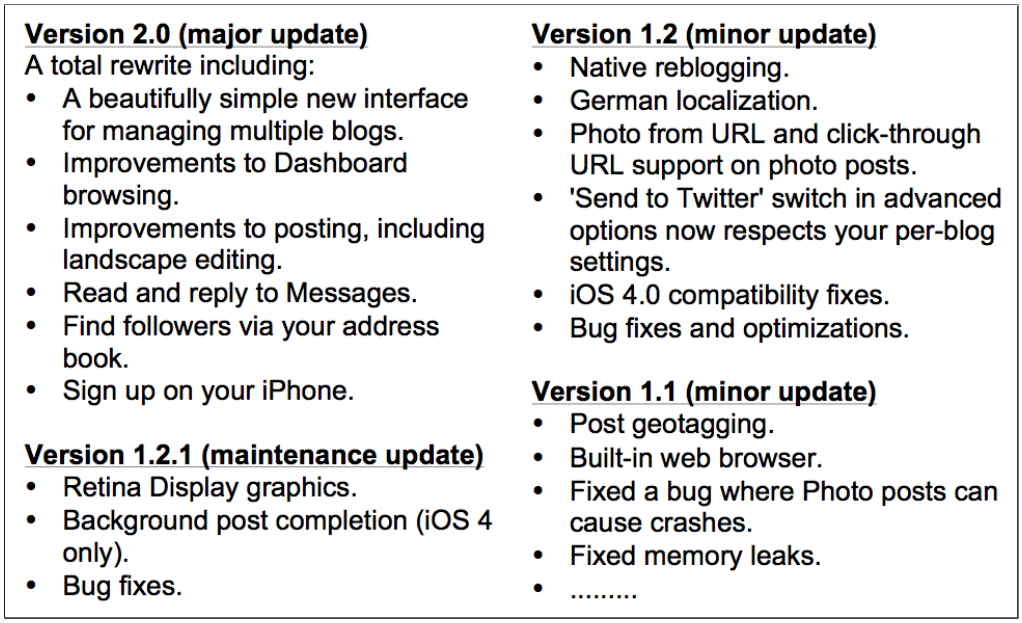


Figure 4.3: An app's changelog chronicles the details of every version update; shown here is an excerpt of the Tumblr app changelog. Version updates typically include new features, enhancements, and/or bug fixes.

4.3.1 Version Features

App versioning is the process of assigning unique version numbers to unique states of the app. Within a given version number category (*e.g.*, *major*, *minor*), these numbers are generally assigned in increasing order and correspond to new developments in the app. Figure 4.3 shows an example of an app's changelog that consists of four different version updates (or *version-snippets*) in reverse order: Versions 2.0, 1.2.1, 1.2, and 1.1. Hereafter, we will use the terms "version-snippet" and "document" interchangeably to refer to the textual description of each version update.

Versions are identified using a conventional numbering structure of "X.Y.Z" where X, Y, and Z represent *major*, *minor*, and *maintenance* categories, respectively:

1. **Major**. Major versions indicate significant changes to the app and is incremented when new major releases are made. This usually denotes

that substantial architectural changes have taken place. For example, in Figure 4.3, Version 2.0 is a *major* version-category.

2. **Minor** — The minor version category is often applied when new functionality is introduced or important bug fixes are introduced. The dependant *maintenance* number (covered next) is reset to zero. For example, in Figure 4.3, Versions 1.1 and 1.2 are *minor* version-categories.

3. **Maintenance** — The maintenance version category is associated with non-breaking bug fixes. For example, in Figure 4.3, Version 1.2.1 is a *maintenance* version-category.

Hereafter, we will use "version-category" as shorthand for version number category.

Besides textual descriptions and version-categories, each version-snippet is also associated with the following information:

1. **Genre Mixture** — Every app is assigned to a subset of pre-defined genres by the developer. For example, the app "Instagram[1]" is assigned to the genres "photo & video" and "social networking." As a version is essentially one of many unique states of an app, the genre mixture in which an app is assigned to is also inherited by its versions. Additionally, as the aforementioned "Instagram" example shows, each app or version is typically assigned to multiple genres[2]. Figure 4.4 shows all of the 40 pre-defined genre labels in the case of Apple's iOS app store (our focus in this study).

2. **Ratings** — It is commonly known that user ratings are directly paired to apps, *i.e.*, user u gives app a a numerical rating r. How-

[1] `http://itunes.apple.com/lookup?id=389801252`

[2] This information (of having more than one genre) is only displayed through the API calls to the app store, and is not displayed in the regular app store that consumers use; instead, only one (primary) genre is shown to the consumer.

Figure 4.4: The 40 pre-defined genre labels on Apple's iOS app store (as of January 2014). The bottom set are gaming sub-genres and only appear on gaming apps.

ever, to be strictly pedantic, the app stores of Apple and Google pair ratings to a particular version of an app: user u gives version v of app a a numerical rating r. Therefore, every version — even if it is the *same* app — receives a different set of ratings from different users. A version that was rated poorly in the past may receive more favorable ratings for later versions.

4.3.2 Generating Latent Topics

In order to find an interpretable and low-dimensional representation of the text in the version-snippets (or documents), we focus on the use of topic modeling algorithms (topic models). A topic model takes a collection of texts as input and discovers a set of "topics" — recurring themes that are discussed in the collection — and the degree to which each document exhibits those topics. We first explore the use of two different topic models: *i*) latent Dirichlet allocation (LDA) (Blei et al., 2003) and *ii*) Labeled-LDA (LLDA) (Ramage et al., 2009), which are unsupervised and semi-supervised topic models, respectively. We also investigate a corpus-enhancing strategy of incorporating version metadata directly into the corpus prior to the

application of topic models. This is to improve the quality of the topic distribution discovered by the topic models.

Modeling Version-snippets with Topic Models

LDA is a well-known generative probabilistic model of a corpus; it generates automatic summaries of latent topics in terms of: *i*) a discrete probability distribution over words for each topic, and further infers *ii*) per-document discrete distributions over topics, which are respectively defined as:

$$p(w|z), \tag{4.2}$$

$$p(z|d), \tag{4.3}$$

where z, d, and w denote the latent topic, the document, and a word, respectively.

However, a limitation of LDA is that it cannot incorporate "observed" information as LDA can only model the text in version descriptions, *i.e.*, LDA is an unsupervised model. In the context of our work, this means that we cannot incorporate the observed version metadata (*e.g.*, version-category and genre mixture) into the latent topics. This leads us to Labeled-LDA (or LLDA), an extension to LDA that allows the modeling of a collection of documents as a mixture of some observed, "labeled" dimensions (Ramage et al., 2009), representing supervision.

LLDA is a supervised model that assumes that each document is annotated with a set of observed labels. It is adapted to account for multi-labeled corpora by putting "topics" in one-to-one correspondence with "labels", and then restricting the sampling of topics for each document to the set of labels that were assigned to the document. In other words, these labels — instead of topics — play a direct role in generating the document's words from per-label distributions over terms. However, LLDA does not

assume the existence of any global latent topics, only the document's distributions over the observed labels and those labels' distributions over words are inferred (Ramage et al., 2011). This makes LLDA a *purely supervised* topic model.

Although LLDA appears to be a supervised topic model initially, depending on the assignment of the set of labels to the documents, it can actually function either as an *unsupervised* or *semi-supervised* topic model. To achieve an *unsupervised* topic model like LDA, we first disregard all the observed labels (if any) in the corpus, and then model K latent topics as labels named "Topic 1" through "Topic K" and assign them to *every* document in the collection. This makes LLDA mathematically identical to traditional LDA with K latent topics (Ramage et al., 2010). On the other hand, to achieve a *semi-supervised* topic model, we first assign *every* document with labels named "Topic 1" through "Topic K" for the *unsupervised* portion, and then use the observed labels[3] (that are unique to each document) for the *supervised* portion.

The *semi-supervised* method of implementing LLDA allows us to quantify broad trends via the latent topics (as in LDA) while at the same time uncover specific trends through labels associated with document metadata. In our work, we treat the version categories and genre mixture as observed labels, and rely on *semi-supervised* LLDA to discover the words that are best associated with the different version-categories and genres, respectively. Similar to LDA, LLDA generates the *topic-word* and *document-topic* distributions in Equations (4.2) and (4.3), respectively, allowing us to obtain the mixture weights of topics for every document. Hereafter, semi-supervised LLDA will be the default LLDA model, and we will also use the terms "topic" and "label" interchangeably.

[3]Note that the number of *observed* labels varies with every document.

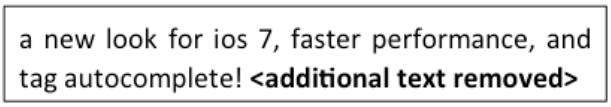
Algorithm 1 How to create "pseudo-terms" from metadata and incorporate them into the corpus (in pseudocode).

```
1: For each doc ‘‘d’’ in corpus:
2: // Note that each doc is a version-snippet.
3:     verText = d.getText();
4:     verCategory = d.getVersionCategory();
5:     // We assume verCategory already has the
6:     // hash-prefix (i.e., ‘‘#’’-prefix).
7:     appId = d.getAppId();
8:     genres = getGenres(appId);
9:     // We assume genres are comma separated values
10:    // and already have the hash-prefix.
11:    verText += genres + ‘‘,’’ + verCategory;
12:    d.setText(verText);
```

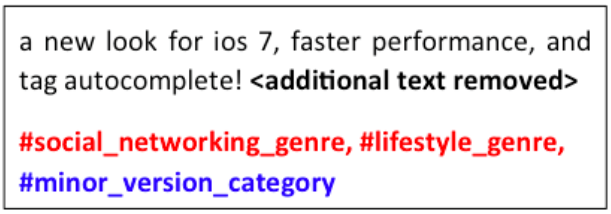
Corpus-enhancement with Pseudo-terms

Aside from employing topic models, we identify another way of incorporating metadata into the latent topics. Inspired by how hashtags are used in Twitter to add content to Twitter messages, we create "pseudo-terms" from the metadata and incorporate them into the set of documents before performing topic modeling. These pseudo-terms can be identified by their "#" prefix (shown in Figure 4.5). Algorithm 1 shows how metadata in the form of pseudo-terms are automatically "injected" into the corpus of version-snippets, as we want to associate these pseudo-terms with the latent topics.

Because LDA and LLDA generate automatic summaries of topics in terms of a discrete probability distribution over words for each topic (Ramage et al., 2009), incorporating pseudo-terms into the corpus allows the topic models to learn the posterior distribution of each pseudo-term (in addition to the natural words) in a document conditioned on the document's set of latent topics. Incorporating these unique pseudo-terms will help in getting topic distributions that are more consistent with the nature of version-snippets. Note that the difference between using the enhanced-



(a) A document **before** "injecting" pseudo-terms.



(b) A document **after** "injecting" pseudo-terms.

Figure 4.5: Metadata such as genre-mixture (in red) and version-category (in blue) are incorporated into documents, which appear in the form of "pseudo-terms" with a "#" prefix.

corpus and the normal corpus is that the former allows both the words and pseudo-terms to be associated with the latent topics, while the latter only allows (natural language) words to be associated with the latent topics. To differentiate between the normal corpus and the enhanced-corpus, we add the prefix "inj"ection to LDA and LLDA; in shorthand, "inj+LDA" and "inj+LLDA," respectively, to denote these approaches.

4.3.3 Identifying Important Latent Topics

We can now model each version-snippet (or document) as a distribution of topics. However, we do not know which topics are important for recommendation. For example, if we knew that users prefer a topic that is related to the promise of high-definition (HD) display support, we would rather recommend an app that includes HD display support in its latest version update over similar apps that do not. Therefore, the importance of each topic differs from app to app, and this is a key contribution of our work.

Furthermore, apps are classified into various genres; each genre works differently to the same type of version update. For example, a version up-

Table 4.1: Genre-topic weighting matrix, where g and z denote a genre and a latent topic, respectively. Every genre-topic pair has a unique weight from weighting scheme. Also, $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}.

Genre	**Latent Topic**						
	z_1	z_2	$\cdots$	z_j	$\cdots$	z_{K-1}	z_K
g_1	$w_{x_{1,1}}$	$w_{x_{1,2}}$	$\cdots$	$w_{x_{1,j}}$	$\cdots$	$w_{x_{1,K-1}}$	$w_{x_{1,K}}$
g_2	$w_{x_{2,1}}$	$w_{x_{2,2}}$	$\cdots$	$w_{x_{2,j}}$	$\cdots$	$w_{x_{2,K-1}}$	$w_{x_{2,K}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
g_i	$w_{x_{i,1}}$	$w_{x_{i,2}}$	$\cdots$	$w_{x_{i,j}}$	$\cdots$	$w_{x_{i,K-1}}$	$w_{x_{i,K}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
g_G	$w_{x_{G,1}}$	$w_{x_{G,2}}$	$\cdots$	$w_{x_{G,j}}$	$\cdots$	$w_{x_{G,K-1}}$	$w_{x_{G,K}}$

date that offers HD display support would be more enticing and relevant on a *game* app instead of a *music* app. Later, in Section 4.5.2, we will show how the inclusion of genre information significantly improves the recommendation accuracy. Because of this, our method includes genre information by default. Table 4.1 shows how we uniquely weight every genre-topic pair with multiplicative weight w_x, where $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}. Note that each genre has a different distribution of importance weights with respect to the set of latent topics.

To compute the weight w, we first introduce a measurement for "popularity" for a document. We use a variant of the popularity measurement detailed in Yin et al. (2012) whereby the popularity is reflected by the votes it receives; as intuitively, the more positive votes it receives, the more popular it is and vice versa. While one may argue that an item receiving a large number of votes (whether they are positive or not) is popular, in this work, we define popular items as those that are "liked" by the majority of the service users, whereby a "like" translates to a rating of 3 and above on the 5-point Likert scale, whereas a "dislike" is a rating of 2 and below.

We formally define the popularity score $\pi(d)$ that outputs a value be-

tween 0 and 1, which factors user ratings into account:

$$\pi(d) = \begin{cases} \frac{pv_d - nv_d}{pv_d + nv_d + 1} & \text{if } pv_d - nv_d > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{4.4}$$

where pv_d and nv_d denote the number of positive and negative ratings of document d, respectively.

We use this popularity score to define the importance weight of a genre-topic pair, w_x:

$$w_x(g, z) = \frac{\sum_{d \in D(g)} p(z|d) \cdot \pi(d)}{\sum_{z' \in Z} \sum_{d \in D(g)} p(z'|d) \cdot \pi(d)}, \tag{4.5}$$

where Z is the set of all K topics, $D(g)$ is the set of all documents that belongs to genre g, $\pi(d)$ is the popularity score of document d, $p(z|d)$ is the *document-to-topic* distribution in Equation (4.3), and $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}. The denominator is used solely for normalization. In other words, w_x is discriminated by the genre, and information from the ratings, along with the distribution of topics, are used to identify its weights.

4.3.4 User Personalization

To incorporate personalization, we need to know each user's preference with respect to the set of latent topics. We determine this importance by analyzing the topics present in the apps that a user u has previously consumed. To compute this factor with respect to a latent topic z, we define the following equation:

$$p(z|u) = \frac{\sum_{d \in D(u)} p(z|d)}{\sum_{z' \in Z} \sum_{d \in D(u)} p(z'|d)}, \tag{4.6}$$

where $p(z|d)$ is the *document-to-topic* distribution defined in Equation (4.3) and $D(u)$ is the set of documents consumed by user u. As in Equation (4.5), the denominator is solely for normalization.

4.3.5 Calculation of the Version-snippet Score

Finally, we calculate the score defined by Equation (4.1). We combine the *document-to-topic* distribution defined in Equation (4.3), the weighting schemes defined by Equation (4.5), the user-personalization factor defined by Equation (4.6), and compute the score as follows:

$$score_x(d, u) = \sum_{z \in Z} p(z|d) \cdot w_x(genre(d), z) \cdot p(z|u), \tag{4.7}$$

where d, u, and z are the document, target user, and latent topic, respectively, $w_x(\cdot)$ denotes the weighting schemes (where $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}), $genre(d)$ is the *genre* of document d, $p(z|d)$ is the *document-to-topic* distribution in Equation (4.3), and $p(z|u)$ is the probability that the target user u prefers topic z. Thus, for each app, we calculate its score based on its latest version to see if it should be recommended.

4.3.6 Combining Version Features with Other Recommendation Techniques

Our work aims at exploring how version features can improve the recommendation accuracy of existing recommendation techniques such as collaborative filtering and content-based filtering. A simple way to integrate version features with the other recommendation techniques is to use a weighted combination scheme, but we also explore a more advanced approach, gradient tree boosting (GTB) (Friedman, 2001), which is a machine learning technique for regression problems that produces a prediction model in the

form of an ensemble of prediction models. We show the results of GTB in our work as it is more superior.

For each of the users, we fit a GTB model to their training data (for each app in the training data that a user has consumed). Each training sample contains the prediction scores of the various recommendation techniques and the actual rating value of the user for the particular app. Note that for our version-sensitive recommendation (VSR) score, we map the score of the version-snippet to the app. We assume a recommendation technique — such as collaborative filtering and content-based filtering or any other — provides a probability of the likelihood of user u consuming or downloading app a. The features given to GTB are a set of probability scores of each of the recommendation techniques, VSR, collaborative filtering, and content-based filtering; the output of GTB is a predicted score between 0 and 5. The predicted ratings are then ranked in reverse order for recommendation.

4.4 Evaluation

We preface our evaluation proper by detailing: *i*) how we constructed our dataset, *ii*) how we chose our evaluation metric, *iii*) our setting for the dataset, and *iv*) the baselines that we compare our approach against.

4.4.1 Dataset

We constructed our dataset by culling from the iTunes App Store[4] and AppAnnie[5]. The dataset consists of the following elements:

1. **App Metadata**. App metadata consists of an app ID, title, description, and genre. The metadata is collected by first getting all the app

[4] `https://itunes.apple.com/us/genre/ios/id36?mt=8`
[5] `https://appannie.com`

IDs from the App Store, and then retrieving the metadata for each app via the iTunes Search API[6].

2. **Version Information**. For each app, we utilize a separate crawler to retrieve all its version information from AppAnnie, which resembles the changelog in Figure 4.3. We treat each app's version as a *document*.

3. **Ratings**. For each version, we utilize yet another crawler to collect its reviews from the iTunes App Store. A review contains an app's ID, its version number, its rating, the reviewer's ID, the subject, and the review comments. This is the source of the rating feature. Note that a rating here is associated to a particular *version* of an app.

We further process the dataset by selecting apps with at least 5 versions, documents (*i.e.*, version-snippets) with at least 10 ratings, and users who rated at least 20 apps. With these criteria enforced, our dataset consists of 9,797 users, 6,524 apps, 109,338 versions, and 1,000,809 ratings. We then perform a 5-fold cross validation, where in each fold, we randomly select 20% of the users as target users to receive recommendations. For each target user, we first remove 25% of their most recent downloaded apps, by default. Additionally, among the training data, 70% is used for training the latent topics while the remaining 30% is used for the training of GTB. Recommendation is evaluated by observing how many masked apps are recovered in the recommendation list.

4.4.2 Evaluation Metric

Our system ranks the recommended apps based on the ranking score. This methodology leads to two possible evaluation metrics: precision and recall.

[6] `https://www.apple.com/itunes/affiliates/resources/documentation/`

However, a missing rating in the training set is ambiguous as it may either mean that the user is not interested in the app, or that the user does not know about the app (*i.e.*, truly missing). This makes it difficult to accurately compute precision (Wang and Blei, 2011). But since the known ratings are true positives, we believe that recall is a more pertinent measure as it only considers the positively rated apps within the top M, namely, a high recall with a lower M will be a better system. As previously done in Wang and Blei (2011) and Chapter 3, we chose Recall@M as our primary evaluation metric.

4.4.3 Optimization of Parameters

For the number of topics K of LDA and LLDA, we experimented on a series of K values between 100 to 1200 for each topic model, and selected the K that maximizes the recall in each model. For the α and β hyperparameters of LDA and LLDA, we used a low α-value of 0.01 as we want to constrain a document to contain only a mixture of a few topics; likewise, we used a low β-value of 0.01 to constrain a topic to contain a mixture of a few words. For the parameters of GTB, we used the default values in scikit-learn[7], whereby we employed 500 trees, a depth level of 3, and the least square for the loss function.

4.4.4 Baselines

We considered two state-of-the-art recommendation techniques as baselines: *i*) probabilistic matrix factorization (PMF) (Salakhutdinov and Mnih, 2008) which represents collaborative filtering (CF); and *ii*) latent Dirichlet allocation (LDA) (Blei and Lafferty, 2009) which represents content-based filtering (CBF).

[7] `https://scikit-learn.org/`

PMF has been widely used in previous works (Agarwal and Chen, 2010; Ma et al., 2011; Salakhutdinov and Mnih, 2008) as an implementation of CF as it is highly flexible and easy to extend. On the other hand, LDA has been used in previous works (Blei and Lafferty, 2009; Lin et al., 2013; Moshfeghi et al., 2011; Wang and Blei, 2011) as an implementation of CBF as it effectively provides an interpretable and low-dimensional representation of the items. Note that in the context of our experiments, LDA's implementation of CBF uses the apps' descriptions as documents — *not* the version features. Besides pure CF and CBF, we also show the recommendation accuracy obtained by hybrid of individual techniques, namely, *i*) CF+CBF, *ii*) CF+VSR, *iii*) CBF+VSR, and *iv*) CF+CBF+VSR, where VSR represents our version-sensitive recommendation approach proposed in Section 4.3.

4.5 Experiments

We first show the recommendation accuracy evaluated with recall by varying the number of latent topics K, and then show how recall is affected when we exclude an app's genre information. After which, we show the performance of the 4 topic models proposed in Section 4.3.2. Finally, we compare our approach with other recommendation techniques, including hybrid methods described in Section 4.4.4.

4.5.1 Recommendation Accuracy Obtained by Different Number of Latent Topics

We optimize the number of topics, K, for our VSR approach with respect to our four new topic models. Figure 4.6 shows the recall when varying K for LDA, inj+LDA, LLDA, and inj+LLDA, respectively. We observe

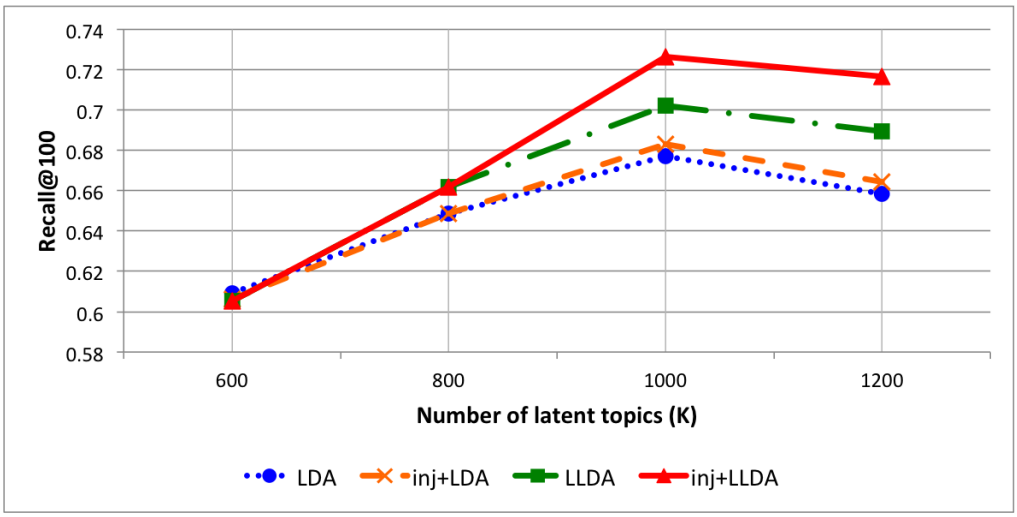


Figure 4.6: For each of the 4 topic models, we experimented with various K between K=100 and K=1200, and show a subsampled chart of K intervals that are fixated at Recall@100.

that K=1000 gives the best recall scores for all four models, and that the recall scores generally show a steep increase towards the optimum (*i.e.*, between K=600 and K=1000), and then gradually decline once K exceeds this optimum (*i.e.*, between K=1000 and K=1200). K=1000 may be seen as a large number of topics, but as observed by Wei and Croft (2006), larger datasets like ours (we have 109,338 documents) may necessitate a larger number of topics to be modeled well. Additionally, as we had previously constrained both hyperparameters of the topic models to be small (resulting in low topic-mixture per document), more topics are needed to represent the set of documents.

4.5.2 Importance of Genre Information

Our framework allows each genre to assign different weights to identical latent topics. In order to determine the importance of genre information, we compare the recommendation accuracies between models with and without genre information. Both variants are based on the best-performing model (inj+LLDA). Figure 4.7 shows that the variant incorporating genre out-

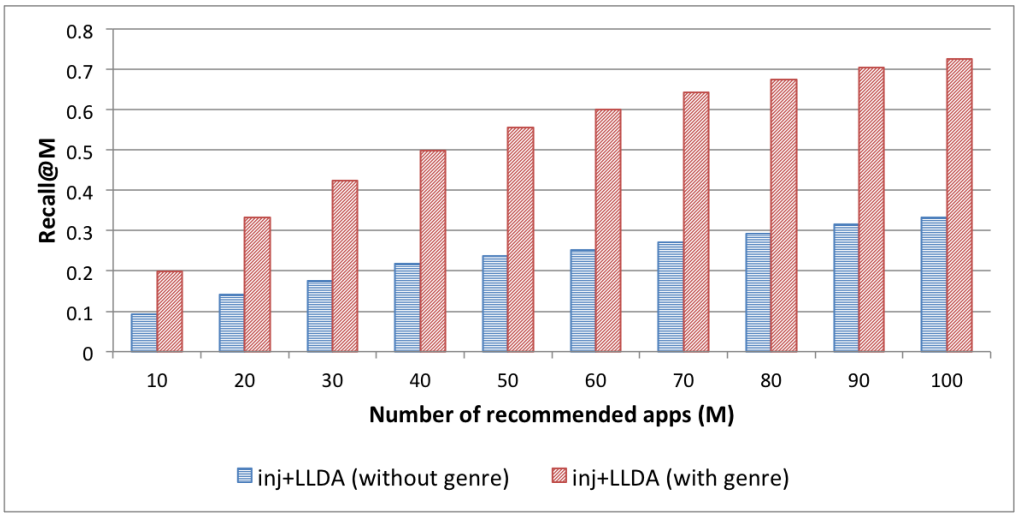


Figure 4.7: Recall scores between the inj+LLDA model that uses genre information and another that does not.

performs the plain model with a statistical significance at $p < 0.01$. We conclude that genre information is an important discriminatory factor, as each genre weights the same type of version update differently. For example, a version update that offers the support for HD displays would be more attractive and relevant to a *game* app instead of a *music* app. Therefore, by discriminating the genres, we assign more relevant weights, which results in better recall. As such, we use genre information in all of the subsequent experiments.

4.5.3 Comparison of Different Topic Models

Figure 4.8 shows the performance of the five different topic model variants: *i*) supervised-LLDA (*i.e.*, without K latent topics), *ii*) LDA, *iii*) inj+LDA, *iv*) LLDA, and *v*) inj+LLDA. So that we can compare unsupervised, supervised, and semi-supervised models, we added supervised-LLDA for the purpose of completeness.

We see that recall is consistently improved as the basic LDA model is incrementally enhanced through inj+LDA, LLDA, and inj+LLDA. Be-

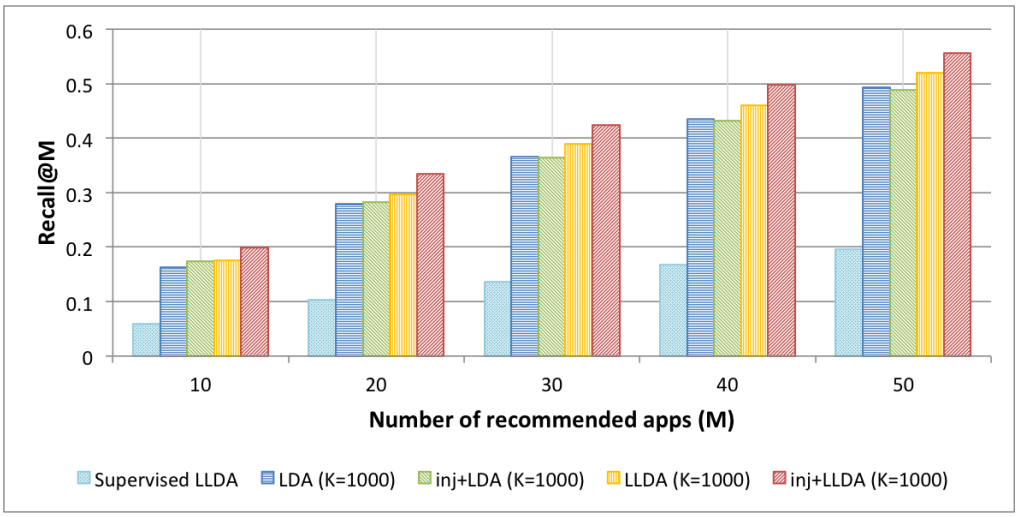


Figure 4.8: Recall scores of different topic modeling schemes with K=1000 as the optimal number of topics.

tween the inj+LDA and inj+LLDA models that use the enhanced-corpus (*cf.* Section 4.3.2) and the LDA and LLDA models that do not, we observe that the enhanced-corpus generally provides better recall, with inj+LLDA showing more significant performance against LLDA. Furthermore, both models of LLDA (*i.e.*, LLDA and inj+LLDA) consistently outperform the pure LDA models, which shows that semi-supervised LLDA models are superior to LDA, which is due to LLDA's ability to quantify broad trends via latent topics while at the same time uncovering specific trends through observed metadata.

We added supervised-LLDA as a baseline for this specific evaluation, but we see that it performs worst among all the baselines. The reason why supervised-LLDA is the worst model despite having "supervision" is that it does not have sufficient topics to properly capture the essence of the corpus. As inj+LLDA is the best-performing model among the topic models we have tested, we use it in subsequent comparisons. We see that use of version metadata improves recall, as the three models that utilize metadata (*i.e.*, inj+LDA, LLDA, and inj+LLDA) consistently outperform the LDA model that only utilizes the text from version-snippets.

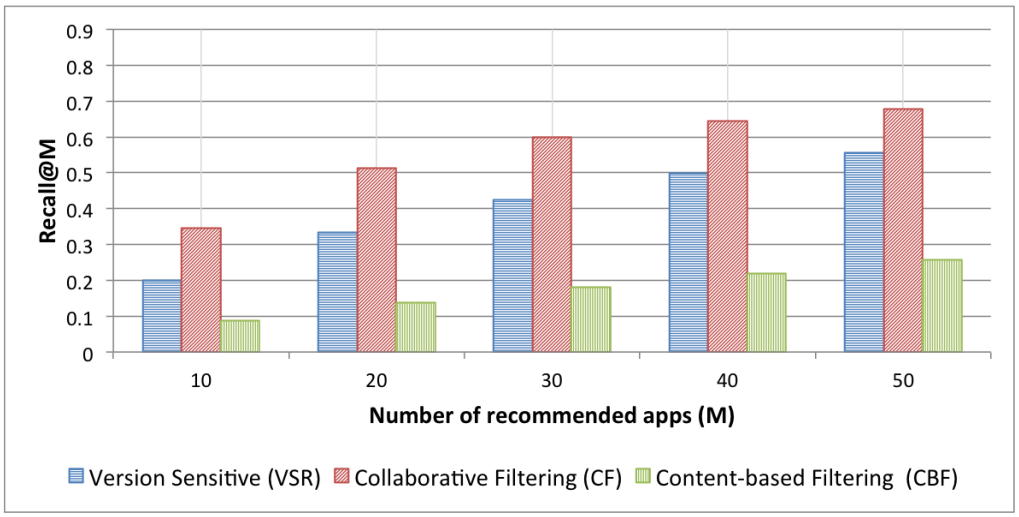


Figure 4.9: Recall scores of our version-sensitive model (VSR) against other individual recommendation techniques.

4.5.4 Comparison Against Other Recommendation Techniques

Figure 4.9 shows the recall scores of the three individual techniques — VSR, CF, and CBF — where the VSR approach uses inj+LLDA at the optimal settings of K=1000. While VSR underperformed against CF, it does outperform CBF. We believe this is because the textual features in the app descriptions are noisy (Lin et al., 2013), resulting in poor recommendation. Thus, among the content-based recommendation approaches of the app domain, version features are promising replacements for app descriptions.

Figure 4.10 shows the combination of individual techniques using GTB. We observe that combining VSR with CBF or CF (*i.e.*, CBF+VSR or CF+VSR) improves both CF or CBF alone. This suggests that version features are a good complement to the traditional recommendation techniques that treat apps as static items. As version features focus on the unique differences between various states of an app, they play a natural complementary role for CF or CBF alone. In addition, we have further

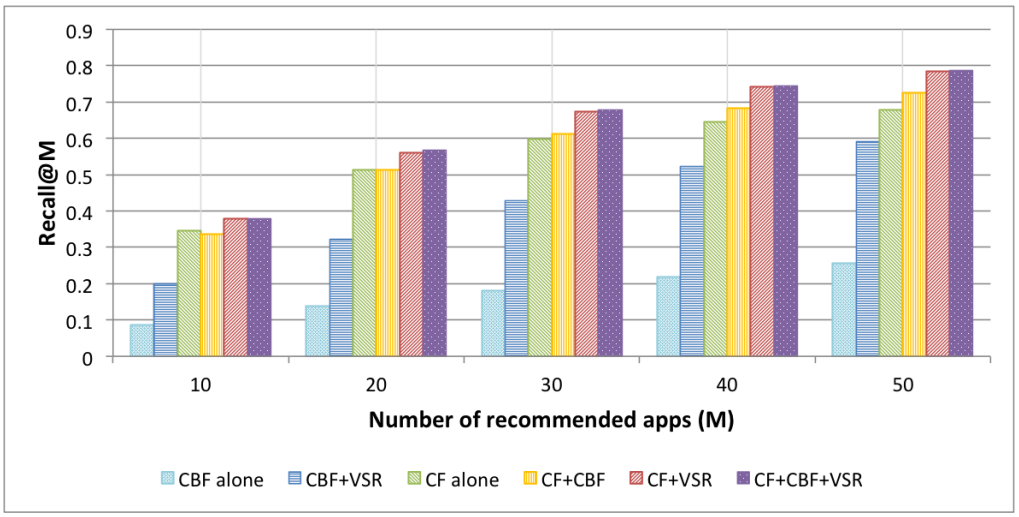


Figure 4.10: Recall scores of various combinations of recommendation techniques.

confirmed that feature-wise, version features are better content descriptions as CF+VSR outperforms CF+CBF. Furthermore, we note that the best performing hybrid is CF+CBF+VSR, though it is roughly on par with CF+VSR. Finally, the hybrid methods CF+VSR and CF+CBF+VSR outperform the pure CF model with a statistical significance of $p < 0.01$ at Recall@50.

4.6 Discussion

We examine the experimental results obtained by the use of version features in detail. First, we perform an in-depth study that compares a recommended version against previous and future versions of the same app. Next, we perform a micro-analysis on individual latent topics and investigate the terms that are found in each topic. Finally, we investigate the effect of injecting more complex version-category information.

4.6.1 Comparison of Previous, Current, and Future Versions of Apps

From our dataset, we only know which version of an app a target user has downloaded. However, we do not know whether the user has or has not seen previous versions of the app before downloading the current version. For example, we only know that Bob downloaded AngryBirds Version 2.1 but we do not know whether:

- Bob had seen previous versions of AngryBirds (*e.g.*, Version 1.0) but was not interested in downloading it at that time, or
- Bob's first encounter with the AngryBirds app was in fact at Version 2.1 and that it was the version that he downloaded.

Hence, we need to consider the situation where a user did not download a target app earlier even though it might be available for download; and that it was only *after* a version-update did the app attract him. For this reason, based on every app that each target user in the training set downloaded, we input the current version (*i.e.*, the version which the target user downloaded) *as well as* the previous and future versions of the same app, and find out whether our system can recommend the exact version that the target user downloaded.

In order to conduct a fair study, we have to take into account the fact that every app has different number of version updates. For example, some apps may only have 5 different version updates while others may have as many as 20 version updates. To solve this problem, we fit the versions of every app into three sets of bins: The first set of bins denotes the previous versions (*i.e.*, bins #1 to #7), the second set of bins denotes the version of the app that a user has downloaded (*i.e.*, bin #8), the last set of bins represent the future versions (*i.e.*, bins #9 to #15). Then, for every app

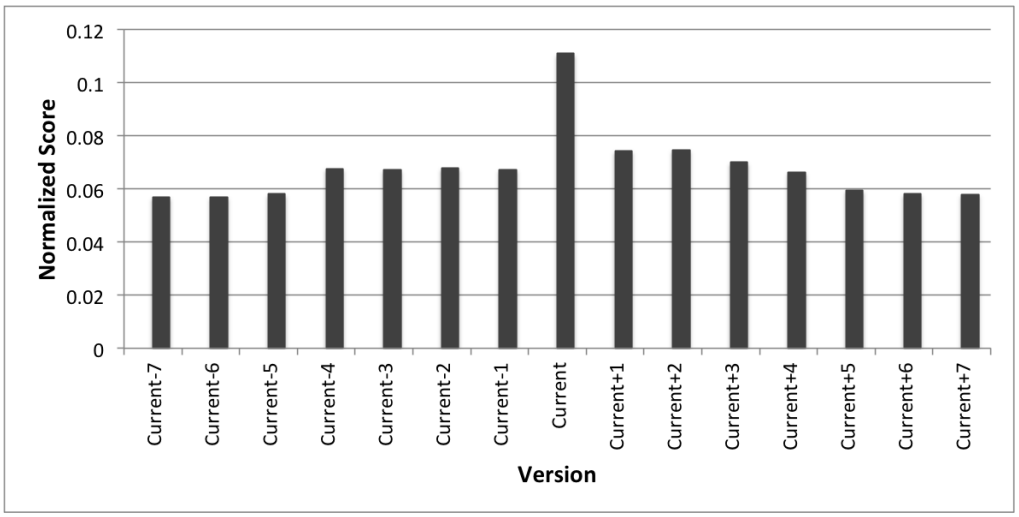


Figure 4.11: Comparison of normalized score among past (current -1 to -7), current, and future (current $+1$ to $+7$) versions.

that a target user has consumed, we calculate the score for each version (explained in Section 4.3.5), and enter the score into the respective bins. Finally, we normalize the score of every bin.

Figure 4.11 shows the normalized score of this analysis for all target users in the training set. We observe that our approach favors the current version (*i.e.*, the one that was downloaded by the target users) the most, thereby indicating that our VSR model effectively targets the version of an app that maximizes its chances of being acquired by the target user. This also reflects that apps tend to go through a series of revisions before being generally favorable; after which the subsequent versions show a decline in general interest, and this suggests the peripheral nature of the subsequent revisions.

4.6.2 Dissecting Specific LDA Topics

To further understand why injecting pseudo-terms into the corpus improves recommendation accuracy, we perform a micro analysis by exploring the latent topics discovered by inj+LLDA. We selected the three most impor-

tant latent topics based on the expectation of each latent topic over the set of training data. Note that each latent topic contains a set of words as well as the injected pseudo-terms.

Figure 4.12 shows the three topics. We observe that every topic coincides with a certain theme. In addition, from the pseudo-terms found in the topic, we can discern the kind of *version-category* and *genre mixture* information the topic belongs to. For example, Topic #385 contains words like "retina" and "resolution", correctly suggesting that the update is display-related. In addition, we observe what genres of apps most likely have such updates, which are the "utilities" and "productivity" apps (in red). Furthermore, we observe that updates in Topic #385 are strongly related to the version-category *minor* (in blue). On the other hand, Topic #47 is associated with "navigation" and "traveling", as the genre-related pseudo-terms (in red) suggests. The top natural language words found in Topic #47 also agree with the hashtags, in that the related updates include improvements in mapping and routing, and that the updates also include alerts and notifications with regards to traveling-related information, such as fuel, points of interests (POIs), and accidents. Finally, as we recall that inj+LLDA allows the incorporation of "observed" labels as topics, the third topic is related to the "medical genre" label and it is closely associated with apps in the neighboring "health & fitness" genre. This "observed" label/topic mainly deals with providing users visual reports (such as graphs and charts) about their personal health (such as periods and pregnancy) as well as the provision of personal tracking and reminders. We observe that the injected pseudo-terms act as a guide for inj+LLDA's inferencing process, which contributes to better latent topic generation. It also helps in understanding the topics further as the metadata (*i.e.*, version-categories and genre mixture) that is imbued in the topics gives users a more comprehensible understanding of the topics.

	Latent Topic #385	Latent Topic #47	Observed Topic “Medical Genre”
Top 20 terms	1. retina: 0.065947 2. display: 0.048744 3. support: 0.046697 4. graphic: 0.034819 5. resolut: 0.029084 6. full: 0.026627 7. #minor_update: 0.024579 8. touch: 0.024579 9. fix: 0.020074 10. ipad: 0.020074 11. #utilities_genre: 0.019664 12. #productivity_genre: 0.018435 13. high: 0.017207 14. optim: 0.016387 15. auto: 0.015159 16. enhanc: 0.014339 17. screen: 0.013111 18. #photo_&_video_genre: 0.013111 19. view: 0.0127 20. mode: 0.012369	1. map: 0.052109 2. #navigation_genre: 0.043457 3. traffic: 0.030958 4. rout: 0.023651 5. improv: 0.023267 6. locat: 0.023011 7. trip: 0.019485 8. #travel_genre: 0.019421 9. road: 0.016857 10. crash: 0.014935 11. address: 0.014294 12. poi: 0.013204 13. fuel: 0.011986 14. auto: 0.011794 15. alert: 0.011602 16. time: 0.011538 17. voic: 0.011153 18. #minor_update: 0.011089 19. weather: 0.010704 20. notif: 0.010128	1. #medical_genre: 0.072711 2. #health_&_fitness_genre: 0.056700 3. pain: 0.032384 4. report: 0.026726 5. medic: 0.026605 6. pregnanc: 0.022512 7. graph: 0.018781 8. period: 0.015290 9. health: 0.015169 10. inform: 0.014326 11. track: 0.012521 12. diari: 0.012400 13. chart: 0.011798 14. drug: 0.011076 15. calcul: 0.010595 16. fertil: 0.009993 17. histori: 0.009752 18. help: 0.008307 19. remind: 0.008187 20. symptom: 0.007706

Figure 4.12: Three most important topics. Each topic shows the top terms, with the inclusive of hashtags. Terms in red are injected terms from genre labels; those in blue, injected terms from version information. Not only does this identify latent topics associated with app updates, it also gives a general overview of the kinds of features found in various version-categories.

Standard Version Categories		
#major	#minor	#maintenance

Advanced Version Categories		
#major	#minor	#maintenance
#minor_after_major	#maintenance_after_major	
#maintenance_after_minor	#major_after_minor	
#major_after_maintenance	#minor_after_maintenance	

Figure 4.13: List of standard and advanced hashtags for corpus-injection.

4.6.3 Importance of Version Categories

To verify the importance of various version-categories (*i.e.*, major, minor, and maintenance), we calculate their respective scores based on *i*) the topic-word distribution from the topic model, and *ii*) the importance score of the latent topics (see Section 4.3.3), which is essentially:

$$\sum_{g \in G} \sum_{z \in Z} w_{\text{inj+LLDA}}(g, z) \cdot p(w = m|z),$$

where m represents one of the strings: "#major", "#minor", or "#maintenance." Note that $p(w|z)$ is the *topic-word* distribution in Equation (4.2). Also note that the equation must be normalized, which results in the score being between 0 and 1.

The importance of each of the three version-categories are as follows: *i*) "#major": 0.128, *ii*) "#minor": 0.656, and *iii*) "#maintenance": 0.216. It is evident that the "minor" version category is the one that is generally more favorable. This is because *major* updates tend to be buggy, while *minor* or *maintenance* updates after a *major* update would likely fix the bugs that occurred in the *major* release, leading to higher user satisfaction. The reason why *minor* performs better than *maintenance* (*i.e.*, 0.656 vs 0.216) is that a *minor* update typically introduces important bug fixes or

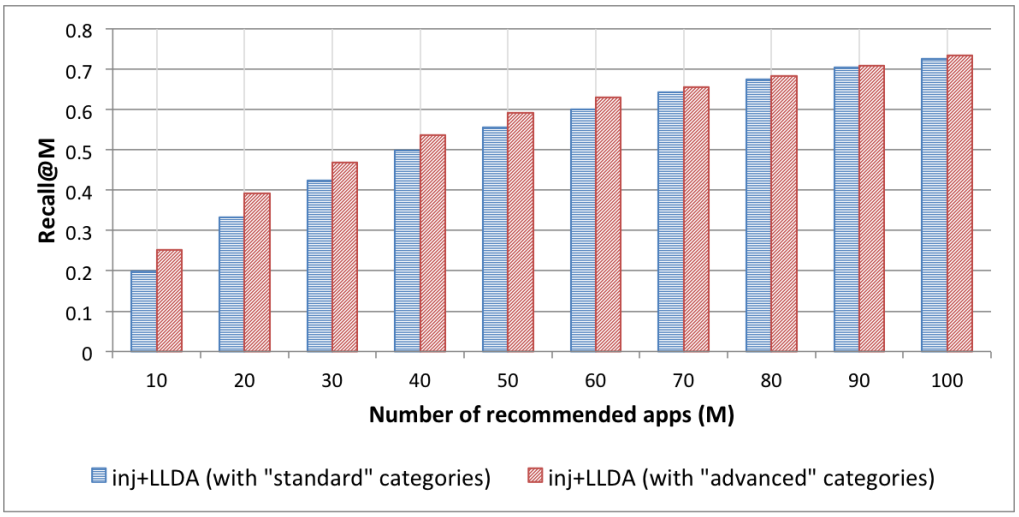


Figure 4.14: Recall scores between the use of "standard" and "advanced" version-categories.

functionalities, which is more appreciable than a *maintenance* update that resolves trivial issues of the app.

As version-categories are valuable features, we hypothesize that the recommendation accuracy can be improved if we further augment the version-categories. More specifically, as we previously only considered three standard version-categories: #major, #minor, and #maintenance, we consider improving the recommendation performance by injecting a more comprehensive list of version-categories into the corpus (as in Figure 4.13). Figure 4.14 shows the comparison between the standard and such an advanced set of version-categories (both models using inj+LLDA). Incorporating the advanced version-categories improves recommendation accuracy, as instead of identifying only 3 standard version-categories, we can discriminate among 6 additional scenarios. The additional details and specifications given by advanced version-categories effectively improve recommendation accuracy. We observe that advanced version-category model outperforms the standard model, particularly at the lower (more important) app recommendation ranks ("M"), although not statistically significantly so.

A more comprehensive modeling of version may be promising, and as such, since there is evidence that the sequence of versions would help, we plan to model the sequence of versions in future work.

4.7 Conclusion

In this chapter, we leverage the unique properties in the app domain and explored the effectiveness of using version features in app recommendation. Our framework utilizes a semi-supervised variant of LDA that accounts for both text and metadata to characterize version features into a set of latent topics. We used genre information to discriminate the topic distributions and obtained a recommendation score for an app's version for a target user. We also showed how to combine our method with existing recommendation techniques. Experimental results show that genre is a key factor in discriminating the topic distribution while pseudo-terms based on version metadata are supplementary. We observed that a hybrid recommender system that incorporates our version-sensitive model statistically outperforms a state-of-the-art collaborative filtering system. This shows that the use of version features complements conventional recommendation techniques that treat apps as static items. We also performed a micro-analysis to show how our method targets particular versions of apps, allowing previously disfavored apps to be recommended.

Chapter 5

A Unifying Framework for App Recommendation

This chapter looks at a unifying framework that combines the recommendation techniques from the previous chapters.

As discussed in the previous chapters, traditional recommendation approaches either learn the preferences of users from their rating history (*i.e.*, collaborative filtering) or through the contents of previously consumed items (*i.e.*, content-based filtering). Although the collaborative filtering approach is used in many recommender domains (*e.g.*, books, movies, music, and apps), its effectiveness is hindered by the lack of sufficient ratings, particularly towards newly released items (*i.e.*, the cold-start problem). In addition, while it may be possible to circumvent the cold-start problem with content-based filtering — since it relies on the textual descriptions which are found in every app's metadata — it is ineffective in the app domain due to noisy and unreliable app descriptions (Lin et al., 2013).

We have demonstrated how to take advantage of the unique properties in the app domain to improve on the recommendation quality. For instance, we can use nascent signals in Twitter to overcome the cold-start problem that besets ratings-based recommendation techniques (see Chapter 3) whereas for situations where there are adequate ratings, we can still improve on collaborative filtering by incorporating the version features of apps as an additional source of useful content (see Chapter 4). However, a limitation of the two state-of-the-art recommendation techniques is that not all apps have Twitter accounts and/or sufficient version information, and this affects the effectiveness of these techniques in the real world.

The fact that different recommendation techniques — both traditional and state-of-the-art — do not perform equally well and are only effective in specific scenarios leads us to pursue the optimal combination of the techniques. That is, given the availability of certain features (*e.g.*, having sufficient ratings, having a Twitter handle, having sufficient version features, etc.), can we come up with a *unifying framework* that marries the strengths of the various app recommendation techniques with respect to the availability of information at a given time?

5.1 A Hypothetical Conceptualization of the App Domain

Inspired by the observation that apps have multiple versions and that each new version is an improvement, we can conceptualize each new version as the *growth* of an app. Therefore, we can view an app's growth as analogous to the developmental growth of a person where there are distinct phases such as *infancy*, *adolescence*, and *adulthood*.

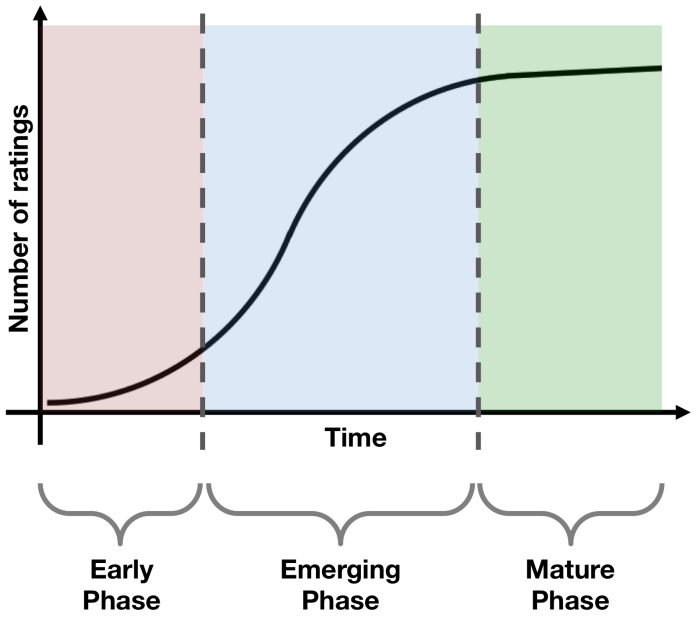


Figure 5.1: Three different hypothetical phases of an app's growth over time: early, emerging, and mature.

Figure 5.1 illustrates the hypothetical idea in which the growth of an app can be categorized into three phases:

1. **Early Phase**. This phase marks the start of an app in which it is introduced in the app store for the very first time. Apps in this phase have few ratings and often face the problems of cold-start and data sparsity. Because of this, recommendation techniques that rely on user ratings such as collaborative filtering are ineffective for apps in this phase.

2. **Emerging Phase**. This phase represents an app's liberation from the cold-start (after it has received a minimum number of ratings) and its growth towards becoming a prominent and mature app. We believe that a mixture of recommendation techniques can be deployed in this phase to make full use of the information available.

3. **Mature Phase**. This phase marks an app's full maturity as it becomes a finished product. Examples of such apps include Angry Birds[1], Flappy Bird[2], and Instagram[3]. We believe that apps in this phase can solely rely on collaborative filtering or some simple and straightforward popularity measurement score.

Not only does each phase require a different set of recommendation techniques, it also depends on the amount of information that is available. For example, if an app has more than 1,000 ratings and has a Twitter page with 20,000 followers, it is likely to be classified as an app in the mature phase. Note that we present our idea about growth phases only as an analogy to motivate the need for a diverse set of strategies for recommendation; our goal is not to identify discrete phases in which recommendation techniques are to be applied, but rather to understand how to make sense of the diverse ecosystem of apps and build a hybrid, unifying framework based on the knowledge obtained.

We now describe our efforts towards developing and evaluating a unifying framework for app recommendation algorithms. Section 5.2 describes the problem and analyzes the various information sources that can be used by the unified recommender system. Section 5.3 describes the unifying framework in detail. Sections 5.4 and 5.5 describe our experimental methodology and results. Finally, Section 5.6 concludes the contributions of this chapter.

[1] "Two billion downloads? We're just getting started, says Angry Birds creator Rovio," Edge, accessed on Jan 30, 2014, `http://www.edge-online.com/features/two-billion-downloads-were-just`.

[2] "What is Flappy Bird? The game taking the App Store by storm," The Telegraph, accessed on Jan 30, 2014, `http://www.telegraph.co.uk/technology/news/10604366/What-is-Flappy-Bird-The-game-taking-the-App-Store-by-storm.html`.

[3] "At 5 Million Users, It's Hard Not To View Instagram Through A Rose-Colored Filter," TechCrunch, accessed on Jan 30, 2014, `http://techcrunch.com/2011/06/13/instagram-five-million-users/`.

5.2 Problem Analysis

5.2.1 Problem Definition

Guided by the hypothetical concept in the previous section, we will explore an objective way to create a hybrid app recommender system that capitalizes on the strengths of each of the previously mentioned recommendation techniques based on the information or metadata that is available. Because different recommendation techniques work in different settings, our goal is to come up with an efficient way to integrate various sources of information into a hybrid model that is able to recommend a set of apps to a target user — regardless of which phase of the app lifecycle it belongs to.

5.2.2 Information for the Unified Model

Inspired by the work of Wang et al. (2012), we classify, on a high level, all information available for the unified system into three distinct groups:

1. the user's history-related information ($\mathbb{H}$),
2. the app's marketing-related metadata ($\mathbb{M}$), and
3. the recommendation scores of different recommender systems ($\mathbb{R}$).

In the unified model, every candidate app's feature vector $\mathbf{x_{u,a}}$ is composed of all three groups of information: $\mathbf{x_{u,a}} = \{\mathbf{x^H_{u,a}}, \mathbf{x^M_a}, \mathbf{x^R_{u,a}}\}$ where $\mathbf{x_{u,a}}$ represents the feature vector of the app a for user u, while $\mathbf{H}$, $\mathbf{M}$, and $\mathbf{R}$ represent the features from the users' history, apps' metadata, and recommendation scores from various recommendation techniques, respectively.

5.2.3 User's History-related Information ($\mathbb{H}$)

User history is essential in building a successful unified system (Wang et al., 2012). This information is primarily extracted from the rating history of

artistId	fileSizeBytes	screenshotUrls
artistName	formattedPrice	sellerName
artistViewUrl	genreIds	sellerUrl
artworkUrl100	genres	supportedDevices
artworkUrl512	ipadScreenshotUrls	trackCensoredName
artworkUrl60	isGameCenterEnabled	trackContentRating
averageUserRating	kind	trackId
averageUserRatingForCurrentVersion	languageCodesISO2A	trackName
bundleId	price	trackViewUrl
contentAdvisoryRating	primaryGenreId	userRatingCount
currency	primaryGenreName	userRatingCountForCurrentVersion
description	releaseDate	version
features	releaseNotes	wrapperType

Figure 5.2: All the components of an app's marketing-related features.

users, and it is a crucial component for profiling users for the purpose of providing personalized recommendations. Each rating history can be summarized into a 3-element tuple: "the rating r that user u gave to app a." The rating history of users is commonly used in personalized recommender systems, including the previous two works in Chapters 3 and 4.

In addition, inspired by how Wang et al. (2012) generate additional user metadata by scrutinizing the genres of items that users have consumed, we also consider the user's general characteristics in each app genre g. For instance, a user might be a loyal consumer of the "games" genre, yet not in the "food & drink" genre. We thus include the number of times (*i.e.*, the count) that apps in genre g were consumed by user u.

5.2.4 App's Marketing-related Metadata ($\mathbb{M}$)

Another important group of information is the app's marketing-related information that is derived from its metadata. Figure 5.2 shows all the components of an app's marketing-related features from the iTunes App Store. Without loss of applicability to other mobile app platforms, the prominent features available include: *i*) description, *ii*) genres, *iii*) developer, *iv*) average user rating, and *v*) user rating count. More details can

be found in Apple's official documentation[4,5]. In addition, we further expand the amount of information by including more segmented data such as version features; we also incorporate external nascent signals in social networks such as Twitter and Facebook.

- **Version features**. As leveraged in Chapter 4, unlike conventional items that are static, apps change and evolve with every revision. It is thus important to include an app's version features into consideration. This information is primarily used in version-sensitive app recommendation.

- **Twitter information**. As leveraged in Chapter 3, some apps have their own Twitter handle. From an app's Twitter handle, we can extract the Twitter-followers that are following the app. This allows us to form a bridge between the mobile app domain and the Twitter domain, and more importantly, gather additional metadata information about an app. A comprehensive Python library for Twitter[6] can be used to retrieve all the required information on Twitter. More information can also be found in Twitter's Developer documentation[7].

- **Facebook Likes**. At the end of Chapter 3, we alluded to other possible social network data sources, particularly Facebook[8]. However, unlike Twitter where we are able to cull individual follower IDs from an app's Twitter account, Facebook does not allow us to retrieve the individual profiles that are following an app's Facebook page (as of Mar 20, 2014). Figure 5.3 shows a chunk of JSON data taken from

[4] "Enterprise Partner Feed Relational," Apple Affiliate Resources, accessed on Mar 15, 2014, `http://www.apple.com/itunes/affiliates/resources/documentation/itunes-enterprise-partner-feed.html`

[5] "Search API," Apple Affiliate Resources, accessed on Mar 15, 2014, `http://www.apple.com/itunes/affiliates/resources/documentation/itunes-store-web-service-search-api.html`

[6] `https://github.com/bear/python-twitter/`

[7] `https://dev.twitter.com/`

[8] `https://developers.facebook.com/docs/graph-api/`

```
{
   "about": "Lead an army of Samurais, Ninjas, and fantastical creatures to ...",
   "category": "App page",
   "company_overview": "Space Ape was founded in July 2012 by a team of industry ...",
   "description": "Samurai Siege is in early Beta Testing and available on ...",
   "is_published": true,
   "talking_about_count": 3533,
   "username": "SamuraiSiege",
   "website": "http://www.spaceapegames.com/ ",
   "were_here_count": 0,
   "id": "399237386851999",
   "name": "Samurai Siege",
   "link": "https://www.facebook.com/SamuraiSiege",
   "likes": 72453,
   "cover": {
      "cover_id": 481302731978797,
      "source": "https://fbcdn-sphotos-d-a.akamaihd.net/1503403_481302731978797_n.jpg",
      "offset_y": 0,
      "offset_x": 0
   }
}
```


Figure 5.3: JSON data from `https://graph.facebook.com/SamuraiSiege`, accessed on Mar 20, 2014.

the official Facebook page of the gaming app "Samurai Siege." Unlike the case of Twitter-followers, we are unable to retrieve any information pertaining to individual Facebook users from Facebook. However, Facebook does provide a *popularity score* which is ubiquitously known as the "Like" button — a feature that allows users to show their support for specific comments, pictures, wall posts, statuses, or fan pages[9]. Figure 5.3 shows that "Samurai Siege" has garnered 72,453 likes (as of Mar 20, 2014). Due to the lack of individual user data on Facebook, we cannot execute the method in Chapter 3 on Facebook as it requires the identification data pertaining to a social network's users. However, based on the "Like" count on Facebook, we can retrieve a popularity score. Although this score lacks the element of personalization, it is still a useful gauge as consumers are more likely to support a brand or item after "liking" it (Harris and Dennis, 2011), which applies to apps as well.

[9] "Facebook's Like button has been lauded as a radically democratic tool allowing users to finally make their opinions heard," The New Inquiry, accessed on Mar 15, 2014, `http://thenewinquiry.com/essays/a-history-of-like/`

5.2.5 Recommendation Scores from Different Recommender Systems ($\mathbb{R}$)

As a unifying framework that integrates various recommendation techniques, it is essential that we include the recommendation scores from the individual recommendation algorithms, namely:

1. **Collaborative Filtering**. We consider a state-of-the-art collaborative filtering recommendation technique, probabilistic matrix factorization (PMF) (Salakhutdinov and Mnih, 2008), to represent collaborative filtering. PMF models the user-item ratings matrix as a product of two lower-rank user and item matrices, and has been used in many previous works (Agarwal and Chen, 2010; Ma et al., 2011; Salakhutdinov and Mnih, 2008) as it is highly flexible and extendable.

2. **Content-based Filtering**. We implement a content-based filtering model using latent Dirichlet allocation (LDA). LDA has been used in previous works (Blei and Lafferty, 2009; Lin et al., 2013; Moshfeghi et al., 2011; Wang and Blei, 2011) as an implementation of content-based filtering as it effectively provides an interpretable and low-dimensional representation of the items. It estimates the score of each item as its total similarity to the user's previous consumption. By using LDA to convert the app descriptions to a latent topic distribution, the similarity of two apps is calculated as the cosine similarity between the product description without stop words and with stemming.

3. **Twitter Information (TWF)**. We use the work mentioned in Chapter 3 to represent the baseline that utilizes Twitter-followers information of apps for the purpose of app recommendation. We abbreviate this recommendation technique as "TWF."

4. **Version-sensitive Recommendation (VSR)**. We use the work mentioned in Chapter 4 to represent the baseline that utilizes the version features of apps for the purpose of providing version-sensitive app recommendation that is aware of the desired functionalities that users are looking for. We abbreviate this recommendation technique as "VSR."

5.3 Unifying Framework

While considerable breakthroughs were achieved by uncovering new features underlying the data from mobile apps, not all apps contain the aforementioned features. As noted by Koren (2009), individual recommendation techniques, even if novel and accurate, are unlikely to make a difference in a practical dataset. A renowned solution, as observed in the Netflix Prize[10], is to combine multiple individual predictors into a single final predictor.

Inspired by BellKor's winning solution for the Netflix Prize (Koren, 2009), we turn to Gradient Tree Boosting (GTB), a machine learning algorithm that iteratively constructs an ensemble of weak decision tree learners through boosting (Friedman, 2001). It produces an accurate and effective off-the-shelf procedure for data mining that can be directly applied to the data without requiring a great deal of time-consuming data preprocessing or careful tuning of the learning procedure (Hastie et al., 2009). It is also a general machine learning algorithm that performs well on learning-to-rank tasks; in particular, it is used as the blender in BellKor's winning Netflix solution (Koren, 2009) as well as the top performing algorithms in the *Learning To Rank Challenge*[11,12] (Wang et al., 2012).

[10] "The Netflix Prize," accessed on Mar 30, 2014, `http://www.netflixprize.com//community/viewtopic.php?id=1537`.

[11] `http://learningtorankchallenge.yahoo.com/workshop.php`

[12] `http://dosen.narotama.ac.id/wp-content/uploads/2012/03/Yahoo-learning-to-rank-challenge-overview.pdf`

GTB provides the following advantages:

- Feature normalization is not required.
- Feature selection is inherently performed during the learning process.
- It is not prone to problems with collinear and/or identical features.
- Models are relatively easy to interpret.
- It is easy to specify different loss functions.

Given the ratings history of users and the metadata of apps, we can generate the feature vector $(\mathbf{x_{u,a}}, r)$ from training data, where $\mathbf{x_{u,a}}$ contains the user and app features as well as the scores from various recommendation techniques, while r is the rating of the user u for app a.

Figure 5.4 shows the details of an exemplar training data $(\mathbf{x_{u,a}}, r)$. The variable r represents the original rating that user u gives app a, whereas the feature vector $\mathbf{x_{u,a}}$ contains three types of features:

1. **User features ($\mathbb{H}$)** in green. This primarily contains the number of times (*i.e.*, count) that an app in genre g is downloaded by the user.
2. **App features ($\mathbb{M}$)** in blue. The features here pertain to the app's metadata or marketing-related information. We include the genres that the app is assigned to, its price, whether or not it is an iPhone or iPad app (can be both), the number of versions it has, the number of Facebook Likes it has (zero if the app has no Facebook handle), the number of Twitter followers it has (zero if the app has no Twitter handle), the number of ratings and the average ratings, whether it is "GameCenter" enabled, and the number of words in its app description.

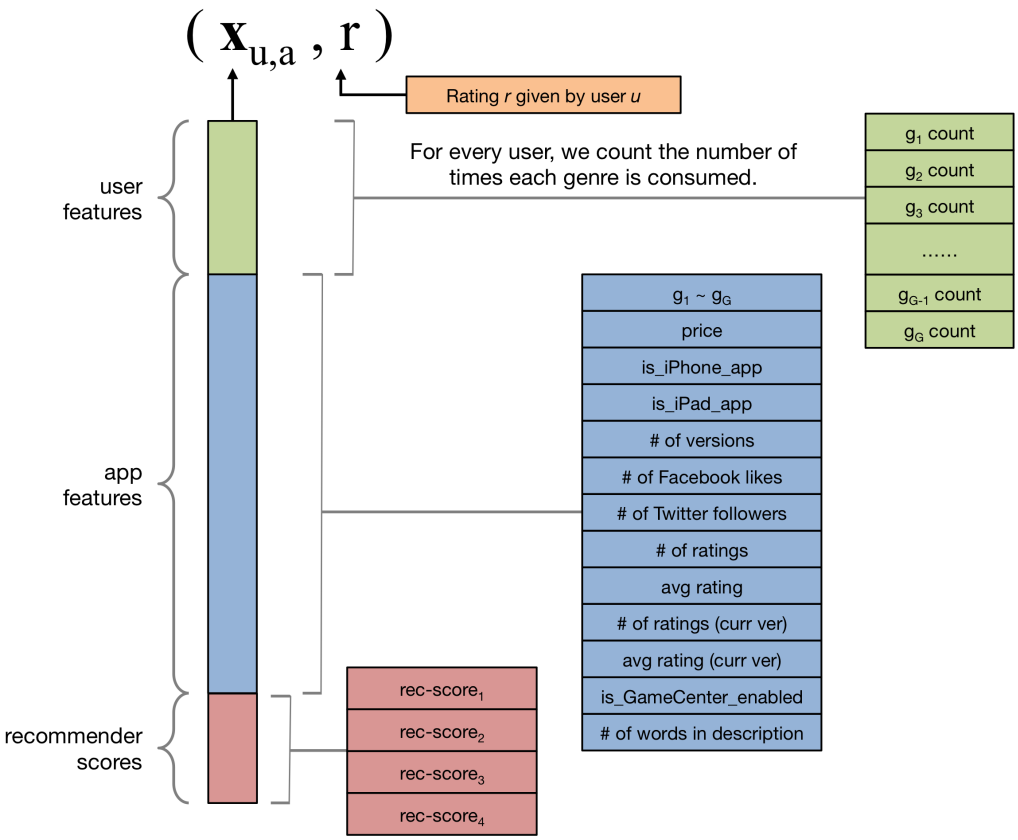


Figure 5.4: Contents in the training data $(\mathbf{x_{u,a}}, r)$, which contains user features, app features, the various recommender scores, and the user's rating.

3. **Recommender scores** ($\mathbb{R}$) in red. We include the scores generated by the various individual recommendation techniques, namely collaborative filtering, content-based filtering, TWF, and VSR.

To generate recommendations, the learned GTB predicts the rating that a user may give to an app. After which, it ranks all recommended apps in descending order of rating to produce a ranked list for each user. Here, we use a popular Python machine learning package from scikit-learn[13] to implement GTB. Friedman (2001) describes more details about the mechanism behind GTB.

[13] `http://scikit-learn.org/stable/modules/ensemble.html`

5.4 Experimental Setup

We create an evaluation dataset based on the information that was collected from Apple's iTunes App Store (app metadata, users, and ratings), App Annie (version information of apps), Twitter (Twitter followers of apps), and Facebook ("Likes" information of apps). In the entire dataset, there are 33,802 apps, 16,450 users, and 3,106,759 ratings after we retain only unique users who have contributed to at least 30 ratings. Among all the 33K apps, 7,124 (21.1%) have Twitter accounts, 9,288 (27.5%) have Facebook accounts, and 10,520 (31.1%) have at least 5 versions. Note that 678 (2%) apps have both Twitter and Facebook accounts.

We take the first 80% of the apps (chronologically) as training data for the individual recommendation techniques, the following 10% is used as the training data for the unified model (*i.e.*, the probe set of GTB), and the last 10% is used for testing. In addition, Figure 5.5 shows the genre distribution of all the apps in the dataset.

5.4.1 Baseline Systems

We compare against several individual recommendation baselines. For basic recommender system solutions, we implemented the four individual algorithms mentioned in Section 5.2.5, namely, *i*) collaborative filtering, *ii*) content-based filtering, *iii*) TWF, and *iv*) VSR.

The hybrid algorithms created by gradient tree boosting (*i.e.*, GTB($\mathbb{R}$), GTB($\mathbb{H}$, $\mathbb{R}$), GTB($\mathbb{M}$, $\mathbb{R}$), and GTB($\mathbb{H}$, $\mathbb{M}$, $\mathbb{R}$)) are our hybrid unifying models with different feature sets, where "$\mathbb{H}$", "$\mathbb{M}$", and "$\mathbb{R}$" represent the various information $\mathbf{x}^{\mathbf{H}}_{\mathbf{u,a}}$, $\mathbf{x}^{\mathbf{M}}_{\mathbf{a}}$, and $\mathbf{x}^{\mathbf{R}}_{\mathbf{u,a}}$ that are mentioned in Section 5.2.2, respectively. For the basic recommender systems, the feature set contains the user's history-related features ($\mathbf{x}^{\mathbf{H}}_{\mathbf{u,a}}$) that are generated from the user's previous ratings history as well as the app data. The hybrid mod-

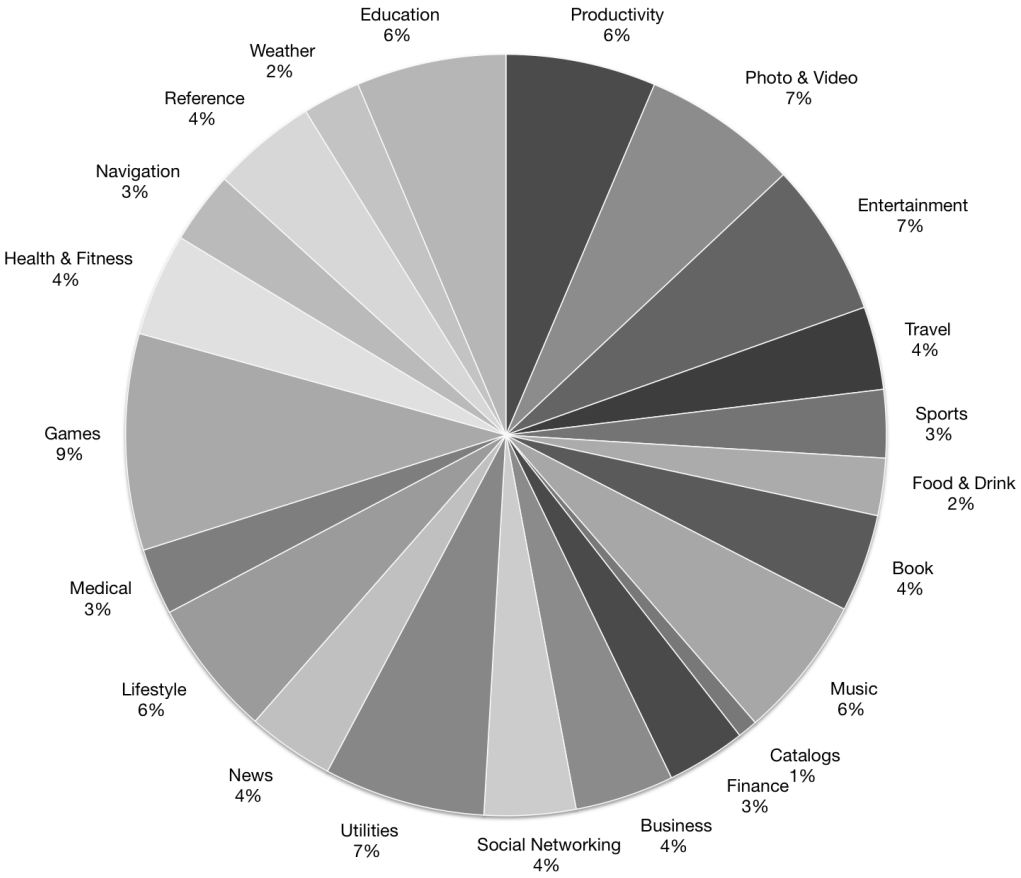


Figure 5.5: Genre distribution of the apps in the dataset.

els further integrate the product's marketing-related metadata ($\mathbf{x}_{\mathbf{a}}^{\mathbf{M}}$) and the recommender scores generated by the individual recommender systems ($\mathbf{x}_{\mathbf{u,a}}^{\mathbf{R}}$). Table 5.1 shows the details of the various recommendation techniques and their feature set.

5.4.2 Evaluation Metric

Our system ranks the recommended apps based on the probability in which a user is likely to download the app. This methodology leads to two possible evaluation metrics: precision and recall. However, a missing rating in the training set is ambiguous as it may either mean that the user is not interested in the app, or that the user does not know about the app (*i.e.*, truly missing). This makes it difficult to accurately compute preci-

Table 5.1: Recommendation techniques studied in the experiments.

Recommendation Technique	**Feature Set**
Probabilistic Matrix Factorization	collaborative filtering with $\mathbf{x_{u,a}} = \{\mathbf{x^H_{u,a}}\}$
Latent Dirichlet Allocation	content-based filtering with $\mathbf{x_{u,a}} = \{\mathbf{x^H_{u,a}}\}$
TWF	Twitter-follower recommender with $\mathbf{x_{u,a}} = \{\mathbf{x^H_{u,a}}\}$
VSR	version-sensitive recommendation with $\mathbf{x_{u,a}} = \{\mathbf{x^H_{u,a}}\}$
GTB($\mathbb{R}$)	$\mathbf{x_{u,a}} = \{\mathbf{x^R_{u,a}}\}$
GTB($\mathbb{H}$,$\mathbb{R}$)	$\mathbf{x_{u,a}} = \{\mathbf{x^H_{u,a}}, \mathbf{x^R_{u,a}}\}$
GTB($\mathbb{M}$,$\mathbb{R}$)	$\mathbf{x_{u,a}} = \{\mathbf{x^M_a}, \mathbf{x^R_{u,a}}\}$
GTB($\mathbb{H}$,$\mathbb{M}$,$\mathbb{R}$)	$\mathbf{x_{u,a}} = \{\mathbf{x^H_{u,a}}, \mathbf{x^M_a}, \mathbf{x^R_{u,a}}\}$

sion (Wang and Blei, 2011). But since the known ratings are true positives, we believe that recall is a more pertinent measure as it only considers the positively rated apps within the top M, namely, a high recall with a lower M will be a better system. As previously done in Wang and Blei (2011) and Chapters 3 and 4, we chose Recall@M as our primary evaluation metric.

5.5 Experimental Results and Analysis

Figure 5.6 shows the comparison of different recommender system's performance for Recall@50. Among the individual recommendation techniques, content-based filtering achieves the best performance, *i.e.*, it outperforms collaborative filtering, TWF, and VSR. It seems, at first, surprising that content-based filtering is the best individual technique among the other individual algorithms, especially against state-of-the-art ones. But given that the dataset contains some apps that: *i*) do not enough ratings for collaborative filtering, *ii*) do not have Twitter accounts (78.9%), and *iii*) do not have sufficient version information (68.9%), it makes sense for these techniques to underperform due to the lack of sufficient information for every app, whereas content-based filtering works better because *all* apps have app descriptions to model with. In other words, in general and practical situations where there are a variety of apps that have *and* do not have

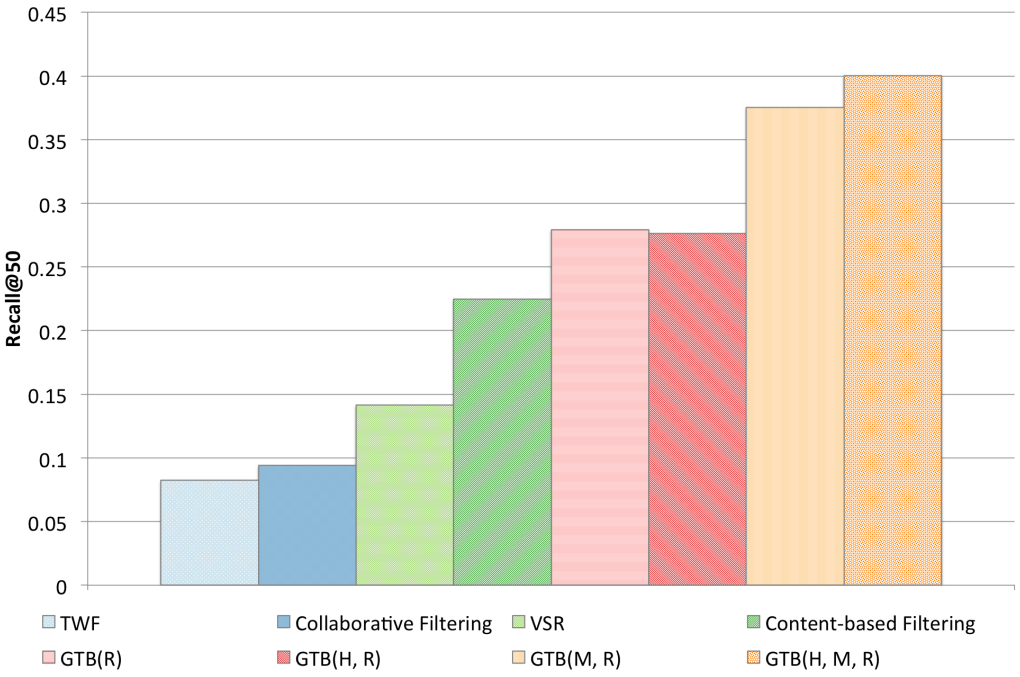


Figure 5.6: Recall@50 obtained by different systems.

ratings, Twitter accounts, and version information, content-based filtering is the more reliable technique.

Next, we explore on the GTB models in Figure 5.6. All of our GTB models outperform the individual techniques. This is expected as other works that use GTB as well (Koren, 2009; Wang et al., 2012) have also reported improvements from individual baselines. We also observe a general improvement in recall when more components are incorporated into the feature set. For example, GTB($\mathbb{M}$,$\mathbb{R}$) outperforms GTB($\mathbb{R}$) while GTB($\mathbb{H}$,$\mathbb{M}$,$\mathbb{R}$) outperforms GTB($\mathbb{M}$,$\mathbb{R}$).

We observe an interesting small anomaly in which GTB($\mathbb{H}$,$\mathbb{R}$) slightly underperforms GTB($\mathbb{R}$), whereas GTB($\mathbb{M}$,$\mathbb{R}$) significantly outperforms both GTB($\mathbb{R}$) and GTB($\mathbb{H}$,$\mathbb{R}$). In other words, the recommendation scores ($\mathbb{R}$) is more useful when it is combined with app metadata ($\mathbb{M}$) than when it is combined with user features ($\mathbb{H}$). This suggests that app metadata ($\mathbb{M}$) complements the feature of recommender scores ($\mathbb{R}$) — which actually makes sense as, given the assortment of app metadata ($\mathbb{M}$) that coincides

with recommendation scores ($\mathbb{R}$), a correlation pattern can be better identified. For example, the app metadata of Twitter followers would complement the recommendation score provided by TWF, while the number of versions would complement the recommendation score generated by VSR; likewise, the number of ratings would complement the recommendation score given by collaborative filtering. On the contrary, as features from user history ($\mathbb{H}$) mainly consists of the count of the number of times a genre is consumed, it has less obvious correlations.

This is not to say that user history ($\mathbb{H}$) has less utility, but rather, the recommender scores ($\mathbb{R}$) benefit much more significantly when they are combined with the app metadata features ($\mathbb{M}$). On the other hand, when user history ($\mathbb{H}$) is combined with both app metadata features ($\mathbb{M}$) and recommender scores ($\mathbb{R}$), we observe an improvement in recall scores, *i.e.*, GTB($\mathbb{H}$,$\mathbb{M}$,$\mathbb{R}$) outperforms GTB($\mathbb{M}$,$\mathbb{R}$), and this demonstrates the utility of incorporating user history features ($\mathbb{H}$).

5.5.1 Ablation Study

The previous experimental results show the overall effectiveness of all four combined recommendation techniques as well as user and app information. To gain a deeper understanding of the individual recommendation techniques, we further perform ablation testing by ablating one of the four individual recommendation techniques from GTB($\mathbb{H}$,$\mathbb{M}$,$\mathbb{R}$), while at the same time, using the user and app metadata, $\mathbf{x}^{\mathbf{H}}_{\mathbf{u,a}}$ and $\mathbf{x}^{\mathbf{M}}_{\mathbf{a}}$.

Table 5.2 shows the ablation study in which we ablate one recommendation technique out of the four. We report several observations from the ablation study:

- Content-based filtering, being the baseline with the best recall score among all individual baselines, also causes the largest dip in recall

Table 5.2: Recall@50 scores in our ablation study.

Feature	**Recall@50**
GTB($\mathbb{H}$, $\mathbb{M}$, $\mathbb{R}$)	**0.403**
GTB($\mathbb{H}$, $\mathbb{M}$, $\mathbb{R}$), excluding TWF	0.363
GTB($\mathbb{H}$, $\mathbb{M}$, $\mathbb{R}$), excluding VSR	0.346
GTB($\mathbb{H}$, $\mathbb{M}$, $\mathbb{R}$), excluding Collaborative Filtering	0.292
GTB($\mathbb{H}$, $\mathbb{M}$, $\mathbb{R}$), excluding Content-based Filtering	0.237
TWF	0.082
VSR	0.141
Collaborative Filtering	0.094
Content-based filtering	0.225

when we ablate it from the unifying model. That is, "GTB($\mathbb{H}$,$\mathbb{M}$,$\mathbb{R}$) excluding content-based filtering" has the lowest score (0.237) among the four ablation baselines. This is unsurprising as it is expected when we omit the strongest individual predictor.

- Although VSR individually outperforms collaborative filtering (0.141 against 0.094), ablating it from the unifying model does not have very much impact; in fact, ablating collaborative filtering has more impact than ablating VSR.

- It would seem that, from this initial ablation study, traditional recommendation techniques such as collaborative filtering and content-based filtering have more utility than VSR and TWF as the two traditional techniques bring about the two biggest dips in recall when we ablate them.

- However, we should not let this relative ablation comparison undermine the improvements that VSR and TWF have brought about. In fact, VSR has led to a 16.5% improvement while TWF has led to

a 11% improvement in recall. More importantly, by utilizing these unique and less obvious signals in the app domain (compared to other traditional domains in recommender systems), we have gained significant improvements for general app recommendation[14]. In other words, different pieces of evidences (*e.g.*, Twitter followers and versions) that, when present, can be utilized sufficiently to create a discernible improvement in recommendation quality.

Still, this initial ablation study does not paint a full picture, especially regarding VSR and TWF, as 68.9% of apps do not have sufficient version information while 78.9% of apps do not have Twitter accounts (see Section 5.4). Therefore, the lack of information does not provide a well grounded conclusion. In order to investigate the real utility of VSR and TWF, we scrutinize our data further by utilizing a subset of data that has sufficient version and Twitter information in the unifying model.

Ablation Study with Sufficient Twitter Information

We perform a near similar study as in Section 5.5.1, but with a dataset with full Twitter information. Table 5.3 shows the recall scores of this study where $\text{GTB}_{\text{TWF}}(\ldots)$ represents the model that uses full Twitter information in our controlled ablation study. We report several observations from this ablation study:

- Under a dataset with full Twitter information, we observe a reordering of recommendation techniques whereby TWF becomes consequential — ablating it causes the largest dip in recall scores (0.338) for the unifying model.

[14]In fact, on 21 September 2009, the grand prize of US$1,000,000 was given to the BellKor's Pragmatic Chaos team which bested Netflix's own algorithm for predicting ratings by 10.06%. That is, US$1M for an improvement of 10.06%.

Table 5.3: Recall@50 scores in our controlled ablation study with sufficient **Twitter information**.

Feature	**Recall@50**
$\text{GTB}_{\text{TWF}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$	**0.446**
$\text{GTB}_{\text{TWF}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding VSR	0.412
$\text{GTB}_{\text{TWF}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding Collaborative Filtering	0.390
$\text{GTB}_{\text{TWF}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding Content-based Filtering	0.386
$\text{GTB}_{\text{TWF}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding TWF	0.338

- Not only does this justify TWF's utility, but more importantly, it illustrates that when certain evidence is available (here, Twitter followers information), they change the signals that are used in the unifying model, allowing TWF to displace the traditional, well-established recommendation techniques.

Ablation Study with Sufficient Version Information

Likewise, we perform a near similar study as in Section 5.5.1, but with a dataset with full version information. Table 5.4 shows the recall scores of this study where $\text{GTB}_{\text{VSR}}(\ldots)$ represents the model that uses full version information in our controlled ablation study. We report several observations from this ablation study:

- Similar to the earlier observation with TWF, under a dataset with full version information, we observe a reordering of recommendation techniques.

- Even though VSR did not displace collaborative filtering in this ablation study, it still resulted in the second largest dip in recall scores (0.344) when we ablate it from the unifying model. In addition, under this dataset, the recall improvement of VSR increased from 16.5% (in Table 5.2) to 22%.

Table 5.4: Recall@50 scores in our controlled ablation study with sufficient **version information**.

Feature	**Recall@50**
$\text{GTB}_{\text{VSR}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$	**0.418**
$\text{GTB}_{\text{VSR}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding TWF	0.396
$\text{GTB}_{\text{VSR}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding Content-based Filtering	0.361
$\text{GTB}_{\text{VSR}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding VSR	0.344
$\text{GTB}_{\text{VSR}}(\mathbb{H}, \mathbb{M}, \mathbb{R})$, excluding Collaborative Filtering	0.335

- This further substantiates that when certain evidence is accessible, they change the way signals are used in the unifying model, which the reordering of recommendation techniques in our ablation study suggests.

The ablation studies on the two controlled datasets (pertaining to full Twitter and version information) clearly illustrate the importance of TWF and VSR in app recommendation, without which we would not have been able to capture Twitter and version signals for the purpose of boosting recommendation quality.

5.5.2 Feature Importance

We further look into each component of the feature set (see Figure 5.4) in the GTB($\mathbb{H}$, $\mathbb{M}$, $\mathbb{R}$) model based on the relative influence[15]. GTB allows us to measure the importance of each component feature. Basically, the more often a feature is used in the split points of a tree, the more important that feature is. Feature importance is important because the input features are seldom equally relevant. Often only a few of them have substantial influence on the response; the vast majority are irrelevant and could just

[15]Friedman (2001) proposed the relative influence for boosted estimates to reflect each feature's contribution of reducing the loss by splitting on the feature.

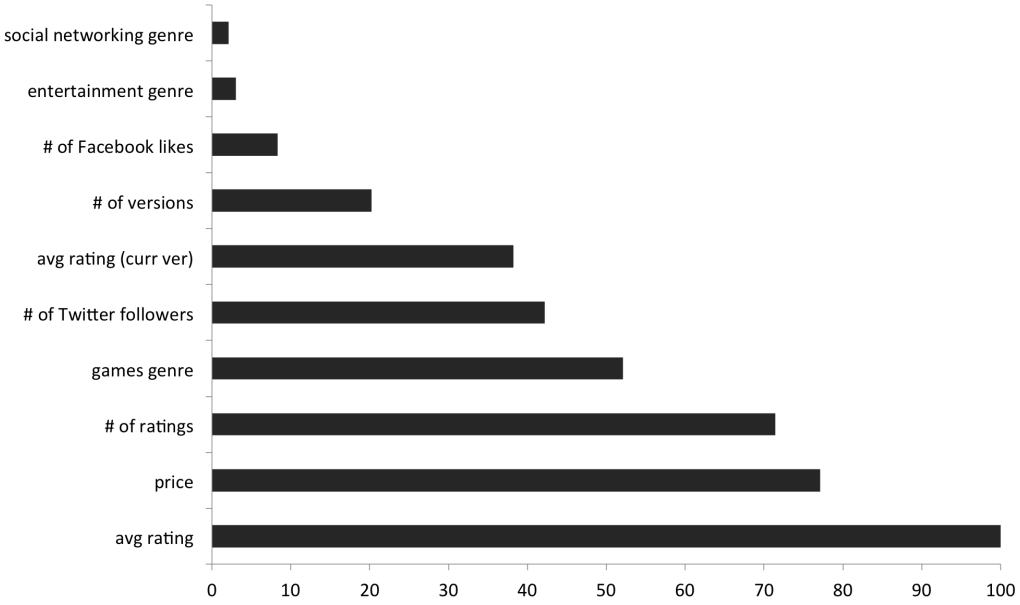


Figure 5.7: Top features in GTB with the highest relative influence.

as well have not been included. Thus, it is often useful to learn the relative importance or contribution of each input feature in predicting the response. Figure 5.7 shows the relative importances of the top features. We observe the following:

- Not surprisingly, the *average rating* is an important factor as, when the average rating is high, there is a natural tendency for the app to be downloaded because of its positive ratings. Therefore, this feature can be used as a strong signal in the unifying framework to make a split in the decision tree. This reasoning is also similar for the *average rating (current version)*.

- *Price* (*i.e.*, free versus paid) is also an important factor, and this evidence coincides with the trend that apps in the app store are heading towards the freemium model[16] — with the proportion of free apps

[16] "Freemium" is a business model by which a proprietary product or service is provided free of charge, but money is charged for advanced features.

taking up 90% of the app store[17,18]. Therefore, the price of an app could be a strong signal for a split in the decision tree.

- The *number of ratings* is also a strong indicator, as the more ratings an app has garnered, the clearer the sign that it is popular and hence, likely to be consumed. It is also a clear sign that the collaborative filtering technique can be employed.

- Not only is the *number of Twitter followers* to the app's Twitter handle an indicator of a strong social reach, the availability of additional Twitter-followers information is also an indicator that our Twitter-followers based recommendation technique (see Chapter 3) can be utilized. Additionally, on a related note, the same reasoning could be used to explain why the *number of Facebook likes* is also one of the top features, as this indicator from Facebook is also a hint of the app's social presence on the popular social networking site.

- The *number of versions* feature also plays an important role as this is a sign that our version-sensitive recommendation technique (see Chapter 4) may be employed. Given that this feature is one of the top features of GTB, it suggests that the version-sensitive recommendation technique is of use here.

- We also notice that some app genres fall under the top features, notably "games," "entertainment," and "social networking" — with "games" having a much more significant influence score. The three genres coincidentally coincide with the findings by Flurry Analytics in Figure 5.8 whereby they discovered that people spend most of

[17] "Paid Apps On The Decline: 90% Of iOS Apps Are Free, Up From 80-84% During 2010-2012, Says Flurry," TechCrunch, accessed on Apr 1, 2014, `http://techcrunch.com/2013/07/18/paid-apps-on-the-decline`.

[18] "It's Over For Paid Apps, With A Few Exceptions," TechCrunch, accessed on Apr 1, 2014, `http://techcrunch.com/2013/10/02/its-over-for-paid-apps`.

their time in apps in the "games," "social networking," and "entertainment" genres across iOS and Android devices.

Finally, we also observe that another set of data from Flurry Analytics, ComScore, and NetMarketShare in Figure 5.9 coincides with our results of the top GTB features in Figure 5.7. For instance, the significant chunks in Figure 5.9 that relate to genres (*i.e.*, "game," "entertainment," and "social messaging") coincide with the same genre labels shown in Figure 5.7. Additionally, the "Facebook" and "Twitter" chunks in Figure 5.9 also coincide with the "# of Facebook likes" and "# of Twitter followers" features in Figure 5.7, which suggests that apps that have a strong presence on these two popular social networks have a tendency to be spotted and subsequently consumed, making them popular candidates to be recommended.

The data from the user studies in Figures 5.8 and 5.9 demonstrate a strong correlation with our GTB feature component analysis in Figure 5.7. It shows how two disciplines (*i.e.*, user studies and GTB feature component analysis) from two different sources of opinions and quantitive angles managed to arrive at the same findings. This further suggests a future direction in mobile app recommendation whereby more focus could be placed in user and trend analysis through social networks — a direction that deviates from traditional research in recommender systems.

5.6 Summary and Contribution

Given that different recommendation techniques work in different settings, we evaluate a method to integrate the various sources of information into a hybrid model that is able to recommend a set of apps to a target user — regardless of whichever scenario or phase. We propose to incorporate the user's prior history, app metadata, and the recommendation scores of various individual recommendation techniques into a unified recommen-

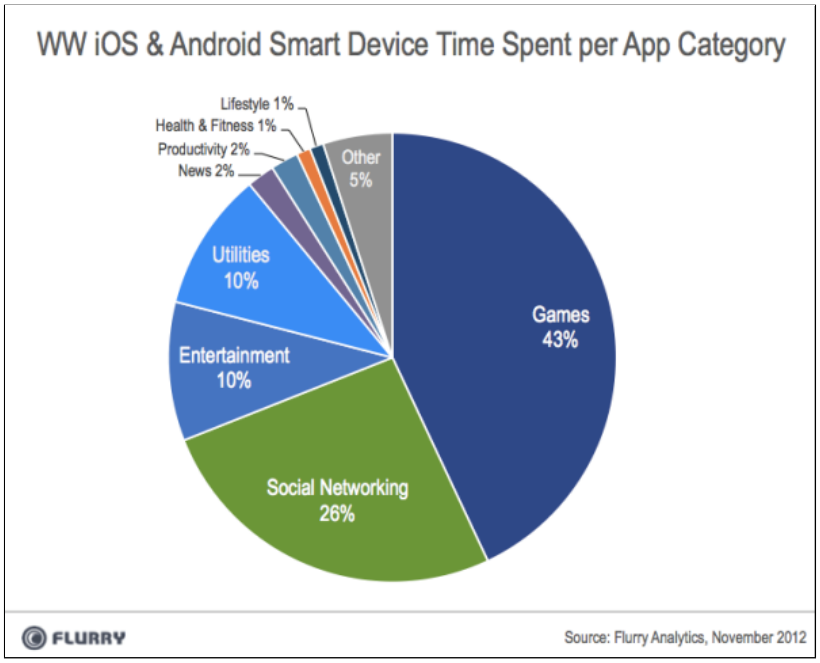


Figure 5.8: Chart showing that 80% of the total time spent is across gaming, social networking and entertainment categories. Source: Flurry Analytics, accessed on Apr 10, 2014, `http://goo.gl/o297Pk`.

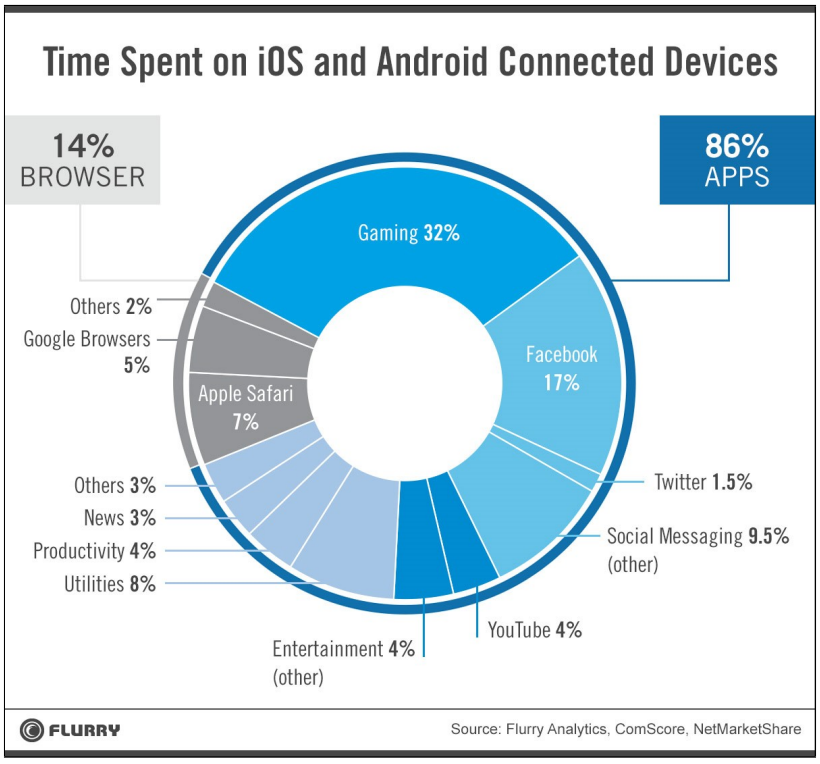


Figure 5.9: Time spent on mobile devices. Source: TechCrunch, accessed on Apr 10, 2014, `http://goo.gl/DLPBl`.

dation model for app recommendation. We then use gradient tree boosting (GTB) as the core of the unifying framework to integrate the recommendation scores by using user and app metadata as additional features for the decision tree. Experimental results show that the unifying framework achieves the best performance against individual and hybrid baselines. We performed a series of in-depth analysis through ablation studies, and demonstrated how different pieces of evidences (such as Twitter and version information) that, when available, could be utilized sufficiently, and how the unifying model dynamically alters the recommendation based on available signals. In addition, we discovered an interesting correlation between important feature components in our unifying framework and user analysis from third-party data analytics companies, which further suggests a future direction in mobile app recommendation where more focus could be placed in user and trend analysis via social networks

Chapter 6

Conclusion and Future Work

This concluding chapter summarizes the work that was done for this thesis, and suggests further research directions that are worth pursuing based on what has been achieved.

The domain of mobile apps is inherently different from other types of digital media (*e.g.*, books, music, and movies). This thesis examines how we can make use of the unique properties of the app domain for the purpose of recommendation. We propose a method that makes use of the nascent information culled from Twitter. The Twitter handle of an app is used to access its Twitter account and extract the IDs of its Twitter-followers. Our method makes use of the data from Twitter-followers to provide recommendations under the cold-start scenario. In addition, we describe another method that makes use of the version features in apps. We show that version features are a possible alternative to app descriptions, and incorporating version features into collaborative filtering helps in recommendation performance. Finally, we provide a framework that factors in the recom-

mendation scores of various recommendation techniques and unifies them into a hybrid app recommendation system.

6.1 Main Contributions

This thesis makes the following contributions to the domain of app recommender systems:

1. Utilize Twitter-followers feature as an alternative source of information to alleviate the cold-start in app recommendation.

2. Utilize version features as an alternative source of content to improve on the quality of existing recommendation techniques.

3. Provide a unifying framework that combines the strengths of conventional and state-of-the-art app recommendation techniques, and perform in-depth analysis of features that uncover interesting connections with data from third-party app analytics.

6.2 Future Work

Research on mobile app recommendation is multidisciplinary. It includes several areas such as data mining, machine learning, personalization, search and filtering, social networks, text processing, and user interaction, among others. Furthermore, current research in recommender systems has strong industry impact, resulting in many practical and potentially successful applications. Still, there are a number of open questions that could be addressed for further research.

6.2.1 Leverage on More Data from Social Networks

We can expand on the use of data from social networks (see Chapter 3). For instance, second-degree relationships such as Twitter-followers *following* the current set of Twitter-followers may be useful, as would using data from n-degree relationships where $n \geq 1$. Likewise, Twitter has auxiliary information that we have not explored, such as *Twitter lists*, which allows users to create a curated group of Twitter users. These curated groups tend to be based on definite themes, such as "Social Good[1]" or "Startups NYC[2]" which can be treated as potential labeled data.

6.2.2 Application of Techniques to Other Domains

We can investigate the effectiveness of the approach in Chapter 3 in other domains, such as music recommendation services. There are many music-related accounts on Twitter. For instance, `@muse`[3] and `@LanaDelRey`[4]. We could follow this process to distill Twitter-followers from these musicians' accounts for the purpose of music recommendation.

6.2.3 Treating versions as Interdependent

The work in Chapter 4 does not take into account the inter-dependency of versions. Hence, more advanced techniques such as treating versions as inter-dependent and using a decaying exponential approach to model how versions are built upon one another in sequence would be interesting.

[1] `https://twitter.com/mashable/lists/social-good`
[2] `https://twitter.com/mashable/lists/startups-nyc-24`
[3] `https://twitter.com/muse`
[4] `https://twitter.com/LanaDelRey`

6.2.4 Exploring Tail Applications

Although solving the problem for tail applications (*i.e.*, unknown or unpopular applications) is not the focus of this thesis, it would be helpful to analyze the distribution of application data on both app stores and social networks, and explore alternatives that target tail applications and tail users.

6.2.5 Exploring Alternatives to Utilize Features

There are alternative methods that could be further explored. For example, one could view recommendation for different genres as different recommendation tasks, and use the multi-task learning (MTL) framework to achieve similar recommendation goals (*i.e.*, different topic importance for different genres). One could also explore simpler approaches, such as converting Twitter and version information into a bag-of-words feature for GTB or bi-linear models. In addition, since *genre* is an important discriminatory component in app recommendation, one could explore using a more granular genre classification scheme into app recommendation techniques, such as the taxonomy scheme from the Interactive Advertising Bureau (IAB) and/or MobileWalla[5].

[5] `https://www.mobilewalla.com/`

Bibliography

Adiya Abisheva, Venkata Rama Kiran Garimella, David Garcia, and Ingmar Weber. Who Watches (and Shares) What on Youtube? And When? Using Twitter to Understand Youtube Viewership. In *Proc. of the 7th ACM International Conference on Web Search and Data Mining (WSDM'14)*, pages 593–602, 2014.

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

Deepak Agarwal and Bee-Chung Chen. fLDA: Matrix Factorization through Latent Dirichlet Allocation. In *Proc. of the 3rd ACM International Conference on Web Search and Data Mining (WSDM'10)*, pages 91–100, 2010.

Charu C. Aggarwal, Joel L. Wolf, Kun-Lung Wu, and Philip S. Yu. Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering. In *Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*, pages 201–212, 1999.

Romel Ayalew. *Consumer Behaviour in Apple's App Store*. PhD thesis, Uppsala University, 2011.

Shumeet Baluja, Rohan Seth, D. Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. Video Suggestion and Discovery for Youtube: Taking Random Walks Through the View Graph. In *Proc. of the 17th International Conference on World Wide Web (WWW'08)*, pages 895–904, 2008.

Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as Classification: Using Social and Content-based Information in Recommendation. In *Proc. of the 15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pages 714–720, 1998.

Punam Bedi, Harmeet Kaur, and Sudeep Marwaha. Trust Based Recommender System for the Semantic Web. In *Proc. of the 20th International Joint Conference on Artifical Intelligence (IJCAI'07)*, pages 2677–2682, 2007.

Nicholas J. Belkin and W. Bruce Croft. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communication of the ACM*, 35(12):29–38, 1992.

Robert Bell, Yehuda Koren, and Chris Volinsky. Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems. In *Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 95–104, 2007.

Upasna Bhandari, Kazunari Sugiyama, Anindya Datta, and Rajni Jindal. Serendipitous Recommendation for Mobile Apps Using Item-Item Similarity Graph. In *Proc. of the 9th Asia Information Retrieval Societies Conference (AIRS'13)*, pages 440–451, 2013.

Daniel Billsus and Michael J. Pazzani. Learning Collaborative Information Filters. In *Proc. of the 15th International Conference on Machine Learning (ICML'98)*, pages 46–54, 1998.

Daniel Billsus and Michael J. Pazzani. User Modeling for Adaptive News Access. *User Modeling and User-Adapted Interaction*, 10(2-3):147–180, 2000.

Daniel Billsus, Michael J. Pazzani, and James Chen. A Learning Agent for Wireless News Access. In *Proc. of the 5th International Conference on Intelligent User Interfaces (IUI'00)*, pages 33–36, 2000.

David M. Blei and John D. Lafferty. Topic Models. *Text mining: Classification, Clustering, and Applications*, 10:71, 2009.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

Philip Bonhard and Martina Angela Sasse. 'Knowing Me, Knowing You' – Using Profiles and Social Networking to Improve Recommender Systems. *BT Technology Journal*, 24(3):84–98, 2006.

John S. Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, pages 43–52, 1998.

Sergey Brin and Lawrence Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. In *Proc. of the 7th International Conference on World Wide Web (WWW'98)*, pages 107–117, 1998.

Robin Burke. Hybrid Web Recommender Systems. In *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 377–408. Springer Berlin Heidelberg, 2007.

Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In *Proc. of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM'10)*, pages 10–17, 2010.

William Cheetham and Joseph Price. Measures of Solution Accuracy in Case-based Reasoning Systems. In *Advances in Case-Based Reasoning*, pages 106–118. Springer, 2004.

Yoon Ho Cho, Jae Kyeong Kim, and Soung Hie Kim. A Personalized Recommender System based on Web Usage Mining and Decision Tree Induction. *Expert Systems with Applications*, 23(3):329–342, 2002.

Christina Christakou and Andreas Stafylopatis. A Hybrid Movie Recommender System Based on Neural Networks. In *Proc. of the 5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*, pages 500–505, 2005.

Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining Content-Based and Collaborative Filters in an Online Newspaper. In *Procs. of ACM SIGIR Workshop on Recommender Systems*, 1999.

Enrique Costa-Montenegro, Ana Belén Barragáns-Martínez, and Marta Rey-López. Which App? A Recommender System of Applications in Markets: Implementation of the Service for Monitoring Users' Interaction. *Expert Systems with Applications: An International Journal*, 39 (10):pages 9367–9375, 2012.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of Recommender Algorithms on Top-n Recommendation Tasks. In *Proc. of the 4th ACM Conference on Recommender Systems (RecSys'10)*, pages 39–46, 2010.

Christoffer Davidsson and Simon Moritz. Utilizing Implicit Feedback and Context to Recommend Mobile Applications from First Use. In *Proc. of the 2011 Workshop on Context-awareness in Retrieval and Recommendation (CaRR'11)*, pages 19–22, 2011.

Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29:1189–1232, 2001.

Jennifer Golbeck. Generating Predictive Movie Recommendations from Trust in Social Networks. In *Proc. of the 4th International Conference on Trust Management (iTrust'06)*, pages 93–104, 2006.

Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm, 2000.

Srinivas Gutta, Kaushal Kurapati, K. P. Lee, Jacquelyn Martino, John Milanski, J. David Schaffer, and John Zimmerman. TV Content Recommender System. In *Proc. of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 1121–1122, 2000.

Lisa Harris and Charles Dennis. Engaging Customers on Facebook: Challenges for e-Retailers. *Journal of Consumer Behaviour*, 10(6):338–346, 2011.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009. URL `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`.

Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 230–237, 1999.

Thomas Hofmann. Collaborative Filtering via Gaussian Probabilistic Latent Semantic Analysis. In *Proc. of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'03)*, pages 259–266, 2003.

Thomas Hofmann. Latent Semantic Models for Collaborative Filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

Thomas Hofmann and Jan Puzicha. Latent Class Models for Collaborative Filtering. In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 688–693, 1999.

Zan Huang, Hsinchun Chen, and Daniel Zeng. Applying Associative Retrieval Techniques to Alleviate the Sparsity Problem in Collaborative Filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1): 116–142, 2004.

Mohsen Jamali and Martin Ester. A Matrix Factorization Technique with Trust Propagation for Recommendation in Social Networks. In *Proc. of the 4th ACM Conference on Recommender Systems (RecSys'10)*, pages 135–142, 2010.

Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato. Fast and Incremental Indexing in Effective and Efficient XML Element Retrieval Systems. In *Proc. of the 14th International Conference on Information Integration and Web-based Applications & Services (iiWAS'12)*, pages 157–166, 2012.

Dohyun Kim and Bong-Jin Yum. Collaborative Filtering Based on Iterative Principal Component Analysis. *Expert Systems with Applications*, 28(4): 823–830, 2005.

Yehuda Koren. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*, pages 426–434, 2008.

Yehuda Koren. The BellKor Solution to the Netflix Grand Prize. *Netflix Prize Documentation*, 2009.

Yehuda Koren and Robert Bell. Advances in Collaborative Filtering. *Recommender Systems Handbook*, pages 145–186, 2011.

Daniel Lemire and Anna Maclachlan. Slope One Predictors for Online Rating-Based Collaborative Filtering. In *Proc. of SIAM Data Mining (SDM'05)*, 2005.

Huizhi Liang, Yue Xu, Dian Tjondronegoro, and Peter Christen. Time-aware Topic Recommendation based on Micro-blogs. In *Proc. of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*, pages 1657–1661, 2012.

Zhung-Xun Liao, Yi-Chin Pan, Wen-Chih Peng, and Po-Ruey Lei. On Mining Mobile Apps Usage Behavior for Predicting Apps Usage in Smartphones. In *Proc. of the 22nd ACM International Conference on Conference on Information & Knowledge Management (CIKM'13)*, pages 609–618, 2013.

Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. Addressing Cold-Start in App Recommendation: Latent User Models Constructed from Twitter Followers. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'13)*, pages 283–292, 2013.

Greg Linden, Brent Smith, and Jeremy York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7:76–80, 2003.

Nathan N. Liu, Xiangrui Meng, Chao Liu, and Qiang Yang. Wisdom of the Better Few: Cold-Start Recommendation via Representative based Rating Elicitation. In *Proc. of the 5th ACM Conference on Recommender Systems (RecSys'11)*, pages 37–44, 2011.

Hao Ma. An Experimental Study on Implicit Social Recommendation. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'13)*, pages 73–82, 2013.

Hao Ma, Haixuan Yang, Michael R. Lyu, and Irwin King. SoRec: Social Recommendation Using Probabilistic Matrix Factorization. In *Proc. of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*, pages 931–940, 2008.

Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. Recommender Systems with Social Regularization. In *Proc. of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*, pages 287–296, 2011.

Bradley N. Miller, Joseph A. Konstan, and John Riedl. PocketLens: Toward a Personal Recommender System. *ACM Transactions on Information Systems (TOIS)*, 22(3):437–476, 2004.

Hemant Misra, Olivier Cappé, and François Yvon. Using LDA to Detect Semantically Incoherent Documents. In *Proc. of the 12th Conference on Computational Natural Language Learning*, pages 41–48, 2008.

Koji Miyahara and Michael J. Pazzani. Collaborative Filtering with the Simple Bayesian Classifier. In *Proc. of the 6th Pacific Rim International Conference on Artificial Intelligence (PRICAI'00)*, pages 679–689, 2000.

Bamshad Mobasher, Xin Jin, and Yanzan Zhou. Semantically Enhanced Collaborative Filtering on the Web. In *Proc. of the First EuropeanWeb Mining Forum (EWMF'03)*, pages 57–76. Springer, 2003.

Raymond J. Mooney and Loriene Roy. Content-based Book Recommending using Learning for Text Categorization. In *Procs. of the 5th ACM Conference on Digital Libraries*, pages 195–204, 2000.

Yashar Moshfeghi, Benjamin Piwowarski, and Joemon M. Jose. Handling Data Sparsity in Collaborative Filtering using Emotion and Semantic-based Features. In *Proc. of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*, pages 625–634, 2011.

Miki Nakagawa and Bamshad Mobasher. A Hybrid Web Personalization Model based on Site Connectivity. In *Proc. of WebKDD Workshop at the ACM SIGKDD International Conference on Knowledge and Discovery and Data Mining*, pages 59–70, 2003.

Daniel Nikovski and Veselin Kulev. Induction of Compact Decision Trees for Personalized Recommendation. In *Proc. of the 2006 ACM Symposium on Applied Computing (SAC'06)*, pages 575–581, 2006.

Seung-Taek Park and Wei Chu. Pairwise Preference Regression for Cold-Start Recommendation. In *Proc. of the 3rd ACM Conference on Recommender Systems (RecSys'09)*, pages 21–28, 2009.

Michael Pazzani and Daniel Billsus. Learning and Revising User Profiles: The Identification ofInteresting Web Sites. *Machine Learning*, 27(3): 313–331, 1997.

Augusto Pucci, Marco Gori, and Marco Maggini. A Random-walk Based Scoring Algorithm Applied to Recommender Engines. In *Proc. of the 8th Knowledge Discovery on the Web International Conference on Advances in Web Mining and Web Usage Analysis (WebKDD'06)*, pages 127–146, 2007.

Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D. Manning. Labeled LDA: A Supervised Topic Model for Credit Attribution in Multi-labeled Corpora. In *Proc. of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP'09)*, pages 248–256, 2009.

Daniel Ramage, Susan Dumais, and Dan Liebling. Characterizing Microblogs with Topic Models. In *Proc. of the 14th International AAAI Conference on Weblogs and Social Media (ICWSM'10)*, pages 130–137, 2010.

Daniel Ramage, Christopher D. Manning, and Susan Dumais. Partially Labeled Topic Models for Interpretable Text Mining. In *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, pages 457–465, 2011.

Raquel Recuero, Ricardo Araujo, and Gabriela Zago. How Does Social Capital Affect Retweets. In *Proc. of the 5th International AAAI Conference on Weblogs and Social Media (ICWSM'11)*, pages 305–312, 2011.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proc. of the 1994 ACM conference on Computer Supported Cooperative Work (CSCW'94)*, pages 175–186, 1994.

Alan Said, Shlomo Berkovsky, and Ernesto W. De Luca. Putting Things in Context: Challenge on Context-Aware Movie Recommendation. In *Proc. of the Workshop on Context-Aware Movie Recommendation*, pages 2–6, 2010.

Ruslan Salakhutdinov and Andriy Mnih. Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo. In *Proc. of the 25th International Conference on Machine Learning (ICML'08)*, pages 880–887, 2008.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. In *Proc. of the 24th International Conference on Machine Learning (ICML'07)*, pages 791–798, 2007.

Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proc. of the 10th International Conference on World Wide Web (WWW'01)*, pages 285–295, 2001.

Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and Metrics for Cold-start Recommendations. In *Proc. of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*, pages 253–260, 2002.

Mark Steyvers and Tom Griffiths. Probabilistic Topic Models. *Handbook of Latent Semantic Analysis*, 427(7):424–440, 2007.

Xiaoyuan Su and Taghi M. Khoshgoftaar. Collaborative Filtering for Multi-class Data Using Belief Nets Algorithms. In *Proc. of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pages 497–504, 2006.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advance in Artificial Intelligence*, 2009:Article No.4, 2009.

Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Matrix Factorization and Neighbor based Algorithms for the Netflix Prize Problem. In *Proc. of the 2nd ACM Conference on Recommender Systems (RecSys'08)*, pages 267–274, 2008.

Slobodan Vucetic and Zoran Obradovic. Collaborative Filtering Using a Regression-Based Approach. *Knowledge and Information Systems*, 7(1): 1–22, 2005.

Chong Wang and David M. Blei. Collaborative Topic Modeling for Recommending Scientific Articles. In *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, pages 448–456, 2011.

Jian Wang and Yi Zhang. Opportunity Model for e-Commerce Recommendation: Right Product; Right Time. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'13)*, pages 303–312, 2013.

Jian Wang, Yi Zhang, and Tao Chen. Unified Recommendation and Search in E-Commerce. In *Information Retrieval Technology*, volume 7675 of *Lecture Notes in Computer Science*, pages 296–305. Springer Berlin Heidelberg, 2012.

Ziqi Wang, Yuwei Tan, and Ming Zhang. Graph-Based Recommendation on Social Networks. In *Proc. of the 2010 12th International Asia-Pacific Web Conference (APWEB'10)*, pages 116–122, 2010.

Xing Wei and W. Bruce Croft. LDA-based Document Models for Ad-hoc Retrieval. In *Proc. of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*, pages 178–185, 2006.

Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. TwitterRank: Finding Topic-sensitive Influential Twitterers. In *Proc. of the 3rd ACM International Conference on Web Search and Data Mining*, pages 261–270, 2010.

Shaomei Wu, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Who Says What to Whom on Twitter. In *Proc. of the 20th International Conference on World Wide Web (WWW'11)*, pages 705–714, 2011.

Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *Proc. of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC'11)*, pages 329–344, 2011.

Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable Collaborative Filtering using Cluster-based Smoothing. In *Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*, pages 114–121, 2005.

Bo Yan and Guanling Chen. AppJoy: Personalized Mobile Application Discovery. In *Proc. of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*, pages 113–126, 2011.

Peifeng Yin, Ping Luo, Min Wang, and Wang-Chien Lee. A Straw Shows Which Way the Wind Blows: Ranking Potentially Popular Items from Early Votes. In *Proc. of the 5th International Conference on Web Search and Data Mining (WSDM'12)*, pages 623–632, 2012.

Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App Recommendation: A Contest between Satisfaction and Temptation. In *Proc. of the 6th International Conference on Web Search and Data Mining (WSDM'13)*, pages 395–404, 2013.

Yi Zhang and Jonathan Koren. Efficient Bayesian Hierarchical User Modeling for Recommendation System. In *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, pages 47–54, 2007.

V. W. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang. Collaborative Filtering Meets Mobile Recommendation: A User-Centered Approach. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, pages 236–241, 2010.

Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. Functional Matrix Factorizations for Cold-Start Recommendation. In *Proc. of the 34th Annual*

International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11), pages 315–324, 2011.

Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Ranking Fraud Detection for Mobile Apps: A Holistic View. In *Proc. of the 22nd ACM International Conference on Conference on Information & Knowledge Management (CIKM'13)*, pages 619–628, 2013.

Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving Recommendation Lists Through Topic Diversification. In *Proc. of the 14th International Conference on World Wide Web (WWW'05)*, pages 22–32, 2005.