NUS
National University of Singapore

School of Computing

Serendipitious Recommendation for Mobile Applications using Item-similarity Graph

Thesis submitted for the degree of

Masters of Technology (CTA)

Dept. of Computer Sciences

Delhi Technological University

2012

Upasna Bhandari

under supervision of

Dr. Anindya Datta (NUS) and Dr.Rajni Jindal (DTU)

I would like to dedicate this thesis to my loving parents, Mrs Geeta Bhandari and Mr Anil Kumar Bhandari. You both are my strength.

Acknowledgements

I would like to extend my deep gratitude towards *Prof. Rajni Jindal*(Supervisor, DTU) for her support and involvement in the successful completion of this work.

My earliest memories of meeting *Prof. Anindya Datta*(Supervisor, NUS) is of being pleasantly surprised by his modest yet inspiring demeanour. He consistently provided me with valuable ideas and suggestions during my research work and has been very considerate all the time. The freedom he gave me in pursing new and sometimes radical ideas taught me how to think and perceive creatively. His continued support and deep involvement have been invaluable during the preparation of this thesis.

I would also like to thank *Prof. Kaushik Dutta* for the in-depth discussions which have been of great help in my research. Special thanks to *Dr. Kazunari Sugiyama* for his valuable suggestions. His patience and modesty is inspiring.

I would like to thank my friends *Nargis Pervin*, *Pooja Roy*, *Sangarlingam Kajanan*, *Feiping Liu* for their friendship and valuable feedback.

I would like to express my love and gratitude towards *Mrs. Geeta Bhandari* and *Mr. Anil Kumar Bhandari* (Parents) who always encouraged me to chase my dreams and supported me in all possible ways to make it this far in life. They fill me with hope and reassurance so that I keep working harder. My brother *Uphar* for being the joy of my life. Someone has become special during last few months and his belief in me cannot go unaddressed, Thank you!. Finally, I am thankful to my spiritual teachers, my grandmothers, *Mrs.Santosh Bhandari* and *Mrs.Raj Vadhera* for being my true heroes.

Contents

List of Figures

Abstract

The domain of mobile applications(*apps*) has recently surfaced and generated lot of interest in academia and industry alike. With an App-store for every leading operating system - Apple, Android, Blackberry, Windows, an explosive growth of Mobile applications is not a surprise. The absolute number of apps currently in existence, as well as their rates of growth, are remarkable. This might be good news for the developers from the revenue perspective but for consumers it means the inherent task of "App Discovery" being intensified.

A reasonable solution to this problem are *Recommender systems*. They usually deal with indicators of user preferences(purchase history/rating history) for suggesting/predicting items for a target user. An effective way to cut-the-queue and straightaway hit the user's interest in shortest possible time, RS are extremely popular with commercial systems today.

To generate relevant recommendations for users, our system tries to leverage the interest patterns in the downloaded applications on mobile phones of users themselves by using item-item similarity graphs. This work essentially tries to overcome the inherent problem of over-specialization in content based recommender system by using graph approach.

This thesis first presents the background literature for recommender systems and then proposes a graph based approach for recommending serendipitous recommendations to a user.

Chapter 1

Introduction

This chapter briefly introduces the research work proposed in the thesis. Section 1.1 gives an overview of recommender systems, what they are and why they are needed. Also it discusses the two most common techniques used in recommender systems- content based and collaborative filtering. Section 1.2 discusses what are the limitations with existing approaches and thus provides motivation for this work. Section 1.3 and 1.4 set the research objective of this work and briefly introduces the proposed approach. Section 1.5 presents an outline of this thesis describing the organization of the remaining chapters. Finally, Section 1.6 gives the summary of this chapter.

1.1 Recommender Systems: What They are and Why are They Needed

Picture this; You walk into a departmental store which today are a dime a dozen intending to buy the usual household items. After you enter the aisle which contains these products, you are faced with an array of choices. Each brand/product is jostling for both shelf space and head space which will enable it to be purchased. The modern consumer is more discerning, however this sense of discrimination is only limited by its ability to differentiate between comprehensible numbers of choice it is presented with. If these choices are increased to manifold the consumer/user is stumped. And herein lies the problem of plenty.

The departmental store problem is limited by the availability of shelf/retail space, but transpose this scenario to the virtual world of retail or "online shopping" as we know today, the space or shelf constraint is eliminated. Now the user is left with what is is page after page of options. So much so that they are overwhelmed. Online retailers or e-commerce websites offer a long list of choices from pin to plane. Any online shopper has one sure shot complaint; there are so many options to choose from, "I can't decide what to buy". Simply put, its the problem of plenty manifesting itself in the virtual world as well.

From the retailer's point of view the problem presents itself differently. It is becoming much more difficult to present the products to the users for enhancing cross-selling and up-selling both. Retailers of all types are now expanding product offerings by adding in-store pickup, free shipping and experimenting with social media. But the inherent problem of "item-discovery" persists.

Since, we all believe in the adage that no two people are the same, recommender systems follow the same approach and would guide the amateur/experienced buyer to navigate through the sea of choices he/she will come across on such websites. Thus recommender systems are tools that help consumers narrow down to a list of potential items of interest or that one final product they would deem fit for purchasing. Let us now look at some of the existing definitions for Recommender Systems in the literature.

Recommender systems are defined as the supporting systems which help users to find information, products, or services (such as books, movies, music, digital products, web sites, and TV programs) by aggregating and analyzing suggestions from other users, which means reviews from various authorities, and user attributes [19][14][13]. Another definition says RSs are software tools and techniques providing suggestions for items to be of use to a user [30][24][9]. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read.

RSs are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially overwhelming number

of alternative items that a website may offer [30]. For example, a book recommender system assists users to select a book to read. In the popular Website, Amazon.com, RS helps to personalize the online store for each customer. Recommendations are thus becoming more and more personalized. Different users, groups, communities receive various suggestions. There exists non personalized recommendation systems as well. For example, in newspapers and magazines, one can often spot *"Top 10 things to do when you are on a holiday"*. But research in recommender system does not deal with such lists. On the other hand, its more directed towards ranking these recommendations. According to the data collected which is a reflection of user's taste and preference, RS tries to rank or score these recommendations and then suggest them to the user. Data from users can either be explicitly collated like ratings/transaction history or implicitly like time spent browsing a particular category of products.

We often ask colleagues or friends for movie/book recommendations, inspired by this simple observation, RS tries to use information about users and suggest products. This approach is termed collaborative-filtering and its rationale is that if the active user agreed in the past with some users, then the other recommendations coming from these similar users should be relevant as well and of interest to the active user.

This sounds rather simple but the magnitude of items present in the market today create a major problem for the users- "How do I find what I need?". Most of the times we do not know for sure as to what we are looking for. Especially with commodities like books/movies, there is a certain kind of emotional setup that guides as to what we prefer reading or watching that day. Thus it is difficult to recommend based on user's preferences in the past alone as to what he would like in the future. But nevertheless the state-of-the-art recommender systems are efficient in producing recommendations which are of potential interest to the user and are accurate according to traditional evaluation metrics like precision and recall.

Providing a user with an exhaustive list of 20 odd recommendations may often leave them overwhelmed and confused. Firstly they do not have a very concrete idea of what they are looking for, secondly it appears like they have been thrown in a pool of options with no rescue in sight. Very often the users complain of a low overall satisfaction. Thus *providing choices is a good thing but*

excess of everything is harmful.

RSs have proved in recent years to be a valuable means for coping with the information overload problem. Ultimately, a RS addresses this phenomenon by pointing a user towards new, not-yet-experienced items that may be relevant to the users current task. Upon a users request, which can be articulated, depending on the recommendation approach, by the users context and need, RSs generate recommendations using various types of knowledge and data about users, the available items, and previous transactions stored in customized databases. The user can then browse the recommendations. She may accept them or not and may provide, immediately or at a next stage, an implicit or explicit feedback. All these user actions and feedbacks can be stored in the recommender database and may be used for generating new recommendations in the next user-system interactions.

In recent years, the research work in recommender systems has dramatically increased. Below are some facts that aid that fact:

- Recommender systems play an important role in highly rated Internet sites such as Amazon[1], YouTube[2], Netflix[3], Yahoo[4], TripAdvisor[5], Last.fm[6], and IMDb[7]. Companies are now developing and deploying RSs as part of their services.

- There are dedicated conferences and workshops related to the field especially ACM Recommender Systems(RecSys), established in 2007 and is now the premier annual event in recommender technology research and applications.

- At institutions of higher education around the world, undergraduate and graduate courses are now dedicated entirely to RSs.

[1]http://amazon.com
[2]http://youtube.com
[3]http://netflix.com
[4]http://yahoo.com
[5]http://tripadvisor.com
[6]http://last.fm.com
[7]http://imdb.com

1.1.1 Data Source and Classification

"Item"-Item is the general term used to denote what the system recommends to users. RS normally focuses on a specific type of item (e.g., CDs, or news) and accordingly its design, its graphical user interface, and the core recommendation technique used to generate the recommendations are all customized to provide useful and effective suggestions for that specific type of item [34]. The value of an item may be positive if the item is useful for the user, or negative if the item is not appropriate and the user made a wrong decision when selecting it. We note that when a user is acquiring an item she will always incur in a cost, which includes the cognitive cost of searching for the item and the real monetary cost eventually paid for the item.

"Users"-Users of RS may have very different goals and characteristics. In order to personalize the recommendations and the human-computer interaction, RSs exploit a range of information about the users. This information can be structured in various ways and again the selection of what information to model depends on the recommendation technique.

"Transactions"-We generically refer to a transaction as a recorded interaction between a user and the RS. Transactions are log-like data that store important information generated during the human-computer interaction and which are useful for the recommendation algorithm that the system employs. For instance, a transaction log may contain a reference to the item selected by the user and a description of the context (e.g., the user goal/query) for that particular recommendation. If available, that transaction may also include an explicit feedback the user has provided, such as the rating for the selected item.

A RS either recommends a list of items for a particular user or predicts the rating of an item for a particular user. For example, the popular recommender system used by movie rental company Netflix predicts the ratings of movies so that user can decide which movie he wants to watch. The online retailer Amazon[1] generates recommendation for a particular user.

RSs are generally classified into collaborative filtering (CF) and content-based filtering (CBF). CF is an information filtering technique based on user's eval-

[1] Amazon

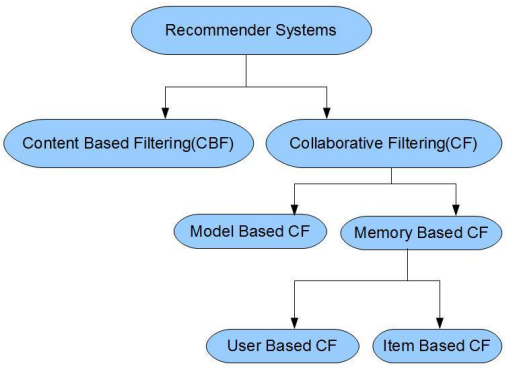


Figure 1.1: Classification of recommender systems

uation of items or previous purchases records but there are two major issues with this approach, namely, sparsity problem and scalability problem. CBF on the other hand, analyzes a set of items rated by an individual user and uses the content of these items, as well as the ratings, to infer profile that can be used to recommend additional item of interest [5]. But a common faced challenge with CBF is that it provides recommendations to be made from a very restricted set of items. Lot of work is currently going on to discover new approaches in recommender systems to solve problems of both CF and CBF.

1.1.2 Content-based Filtering for Recommendations

The content based filtering approach consists of analyzing the content of the items being recommended. Each user is treated individually. It is a purely item-based; there is no sense of community or group. Items of target users are compared to items purchased by other users and then recommendations are made. It is highly possible that items are recommended from a small subset.

1.1.3 Collaborative Filtering for Recommendations

It is the most popular approach for recommendations currently used by Amazon, YouTube etc. Collaborative filtering basically works on the principle of word-of-mouth recommendations. Herlocker et. al [17] states that "one of the most successful technologies for recommender systems is called collaborative

filtering". This system is based on the collection of taste information from many users. But there is a sense of a community. It assumes that a group of users will have a similar liking to items then tries to find the probability of unpredicted items for an active user based on a linear weighted combination of other user's preferences.

1.2 Research Motivation

1.2.1 Limitations with Existing Approaches

One of the most inherent problem with recommender systems is the problem of *"Item Discovery"*. With items being added every day in the market, it is getting more and more difficult for users to reach out to good items which are potentially relevant to their interest. This issue in the domain of mobile applications gets more and more intensified. Mobile applications are popularly known as "Apps". Now app market is huge. Nearly 1.1 million of free and paid apps are available now in leading app stores like Apple,Android,Windows,Blackberry. There are about 100 movies and 250 books that get released per week as compared to the release of around 15,000 apps per week.This means that the problem of item discovery poses serious threat to the recommendation generation. Thus there is a need to develop techniques that can get the users to experience useful products without wasting much time. Other issues with existing approaches are listed as follows:

- Scalability: The issue of scalability has long been associated with RS. Algorithms that are sufficient in one scenario become insufficient in the other. With increasing number of products and users, scalability issues of transactional data have become even worse and need to be taken care of.

- Proactive recommender systems: There is a rising need for such recommender systems which do not require explicit inputs to produce recommendations. They should be able to predict when and what to recommend.

- User's privacy: There is a rising insecurity amongst users since most RS consume user data to make predictions. These systems need to ensure that they use the data in a justified and sensible manner.
- Diversity in a recommendation list: Everyone is now waking up to the idea of moving beyond the "obvious". The recommendations made by state-of-the-art are far too obvious and offer very little diversification in terms of the choices it presents to the users. RS now needs techniques to incorporate surprising recommendations to increase the diversity of the recommendation lists.

1.2.2 What is Serendipity and its Scope in Content Based and Collaborative Filtering Approaches

Serendipity means a "happy accident" or "pleasant surprise"; specifically, the accident of finding something good or useful without looking for it[37]. We often we find things or meet someone where we least expected, although a surprise, it is a happy surprise. In case of recommendations we call them Serendipitous Recommendations. Serendipitous recommendations are recommendations targeted for a particular user that might be of his interest even when he has not explicitly displayed interested for this item.

Collaborative filtering have been successful in providing serendipitous recommendations. In item-based methods, the rating predicted for an item is based on the ratings given to similar items. Consequently, recommender systems using this approach will tend to recommend to a user items that are related to those usually appreciated by this user. For instance, in a movie recommendation application, movies having the same genre, actors or director as those highly rated by the user are likely to be recommended. While this may lead to relevant recommendations, it does less to help the user discover different types of items that he might like as much. Because they work with user similarity, on the other hand, user-based approaches are more likely to make serendipitous recommendations. This is particularly true if the recommendation is made with a small number of nearest neighbors. For example, a user A that has watched only comedies may be very similar to a user B only by the ratings

made on such movies. However, if B is fond of a movie in a different genre, this movie may be recommended to A through his similarity with B.

User Based	Item Based
More popular Input: Set of users A={a1,a2,a3...) Set of products B={b1,b2,b3...) **Step1: Neighborhood formation** ai-active user C(ai,aj) based on ri,rj(pearson or cosine) Top most similar users-**clique(ai) $\subseteq$ A.** **Step2: Rating Prediction** All the products bk that ai's neighbors aj $\in$ clique(ai) have rated and which are new to ai,wi(b_k) i.e., ri(bk) = $\perp$, a prediction of liking wi(bk) is produced.	**Similarity function** C: B×B-->[-1,+1] 2 items b_k and b_e similar i.e C(b_k,b_e) increases if user who rates one tends to rate other as well **Step1: Neighborhood formation** **clique(b_k) $\subseteq$ B** **Step2: Rating Prediction** $w_i(b_k) = \frac{\sum_{b_e \in B'_k} (c(b_k, b_e) \cdot r_i(b_e))}{\sum_{b_e \in B'_k} \lvert c(b_k, b_e) \rvert}$, $B'_k := \{b_e \mid b_e \in \text{clique}(b_k) \wedge r_i(b_k) \neq \perp\}$

Figure 1.2: Comparison of user-based and item-based collaborative filtering recommendation techniques

Content based filtering tends to make rather relevant predictions and are not desirable for making serendipitous recommendations. What do we mean by this? Well these predictions are rather restricted in terms of the options it presents to the user. Since the method is primarily driven by information like user's past preferences towards certain items, it tends to recommend items very similar to them. But this might not be completely desirable. Very often we are pleasantly surprised by recommendations from friends or family that are nowhere close to our past preferences. Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This

drawback is also called serendipity problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty. To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be recommended just that kind of movies. A perfect content-based technique would rarely find anything novel, limiting the range of applications for which it would be useful [34].

This leads us to research objective of this work, can content based recommendation be elevated to make serendipitous recommendations? The next section addresses this concern.

1.3 Research Objective

1.3.1 Can Serendipitous Recommendations be Generated More Effectively Using Graph Based Recommendations?

Having established the fact that recommendations are necessary and there will always be a need to provide users with suggestions especially due to the information overload, we will now look how serendipity in recommendation scan be achieved.

Usually collaborative filtering approaches are used to make serendipitous recommendations. It is quite understandable since collaborative filtering framework allows far more opportunities for surprise recommendations since it first finds similar users to a target user and then recommends new items. Content based approach relies on items that user has liked in the past and the item description information and thus are not used to make serendipitous recommendations because of its shortcoming- ”over-specialization”.

With the over abundance of mobile apps, it is often difficult for consumers to get the apps of their interest along with considerable novelty.I understand this is nothing but ”stereotyping” a user because the of the choices he made in his past.Recommendations drawn from the past history of user’s transactions or ratings often tends to give very obvious choices which are highly accurate but does not work well to enhance the user’s overall experience. Moreover novel and serendipitous recommendations are necessary to broaden user’s view in the recommendation flow [18].

In this work, we try to use graph-based recommendations which can address the issue of "over-specialization" associated with content based methods to present the users with serendipitous recommendations. Graph approaches for recommendations mainly concentrate on using the inherent graph structure to predict proximity between users to measure similarity between two users [12][23]. They have also been used to predict rating of unrated items by using ratings for items connected to it. For our domain(i.e., mobile apps) we try to effectively represent a user on the product-product similarity graph by using information like apps downloaded on his mobile phone, the frequency of usage,duration of usage. Later using graph theory operations we try to find interest patterns inherent in a graph structure. By controlling the similarity threshold between apps, we introduce serendipity into our recommendation process and suggest beyond the obvious but still being novel to the user's preference.

1.4 Proposed approach

1.4.1 Graph Based Recommendations to Address Over-Specialization in Content Based Filtering

Though this work is different from content based and collaborative filtering, the proposed approach still manages to fit into a standard recommendation framework. The proposed method has the following steps:

- App-App Similarity Graph Construction- First the similarity,$Sim(App_i, App_j)$ between two mobile apps is calculated by using metadata like title, category, description. Standard TF-IDF score is used because of its effectiveness in calculating similarity between short texts. This information is thus used to construct a $G = (V, E)$ where V is the set of vertices(apps) and E is set of edges between them.

- User Preferences on the graph- Information available on the user's mobile phone like the applications already installed, frequency of the usage, duration of the usage etc. can be encashed upon by considering it as a holistic view of a user's interest span.

- Recommendation prediction- For generating recommendations, we find paths between pairs of apps following edges with a similarity score higher than a threshold. We do not intend to follow very high similarity score since it would lead to less serendipitous recommendations.

This approach will help overcome the most prominent problem existent in content-based filtering, personalization or over-specialization. The recommendation prediction does not require ratings of a product for it to appear in the recommendation lists. By collecting information from user's mobile phone we concentrate on what his current preferences are instead of relying on his past preferences. Recommendations thus generated are highly serendipitous.

Note that we do not take any explicit input from the user in the form of reviews or rankings instead we simply focus on what is available on the user's phone to serve as the window to their taste set as far as mobile apps are concerned. This is slightly different from popular RSs based on collaborative filtering which rely on ratings and reviews to compare user profiles and item profiles. The user provides no input and recommendations are simply generated by collecting data from the user's phone.

1.5 Organization of Thesis

This thesis is structured into five chapters.It is organized as follows:

Chapter 2 provides the essential background and context for this thesis and provides a complete justification for the research work described in this thesis.

Chapter 3 provides the literature review of existing work in the domain of serendipitous recommendations and the important aspects of generating serendipitous recommendations.

Chapter 4 provides the details of the proposed method and outlines the graph based recommendation system.

Chapter 5 describes the experimental results.

Chapter 6 presents conclusion and future work.

1.6 Summary

This chapter has laid the foundations for this thesis. It briefly introduced the research problem, research objectives and the proposed solution framework. A justification for the research problem is outlined, together with an explanation of the research methodology used. The next chapter examines the pertinent literature most relevant to this research.

Chapter 2

Basic Issues with Recommender Systems

This chapter provides background literature for the research work proposed in the thesis. Section 2.1 and 2.2 talk about content based filtering in detail, their advantage and shortcomings. Sections 2.3 and 2.4 discuss collaborative filtering and what are their limitations and advantages. Section 2.5 talks about neighbourhood based recommender system which is another popular technique. Finally section 2.6 talks about graph based recommendations which is used in the proposed approach.

Items cannot be recommended unless they are worth recommending. In other words, we need to find out the utility of an item and thus this task is at the heart of a recommender systems. Ranking or scoring is the most common and easiest way to calculate this utility. This also helps in comparing two items so that one can be chosen over another.

Content-based filtering: The system learns to recommend items that are similar to the ones that the user liked in the past. The similarity of items is calculated based on the features associated with the compared items. For example, if a user has positively rated a movie that belongs to the comedy genre, then the system can learn to recommend other movies from this genre

Collaborative filtering: The simplest and original implementation of this approach recommends to the active user the items that other users with similar

tastes liked in the past. The similarity in taste of two users is calculated based on the similarity in the rating history of the users. Collaborative filtering is considered to be the most popular and widely implemented technique in RS.

2.1 Content-based Filtering

The basic idea of content based recommender systems is to recommend items that a user has liked in the past. So the idea is to use the features or attributes of items liked by a user and compare those with other items and recommend similar items. The idea of personalization for a user is very obvious in this context. Content based filtering as discussed above recommends items in the basis of prior preferences of a user.

RSs implementing content based approach have to be involved in computing similarities between documents or textual descriptions previously used or rated by the user. This in a way represents the user on the basis of his prior preferences.

As shown in 2.1, basic components of a Content based filtering stem are as follows [34].

(1) CONTENT ANALYZER-The main role of the content analyzer is to structure the information provided to the recommender systems. It could be in the form of documents, web pages, descriptions etc. This information usually gets converted into feature vectors where each document, web page, textual description is represented as a vector in the target space. This becomes an input to the next stage.

(2) PROFILE LEARNER- This module collects data representative of the user preferences and tries to generalize this data, in order to construct the user profile. Usually, the generalization strategy is realized through machine learning techniques [27], which are able to infer a model of user interests starting from items liked or disliked in the past

(3) FILTERING COMPONENT - This module exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. The result is a binary or continuous relevance

judgment computed using some similarity of potentially interesting and is realized by computing the item vectors.

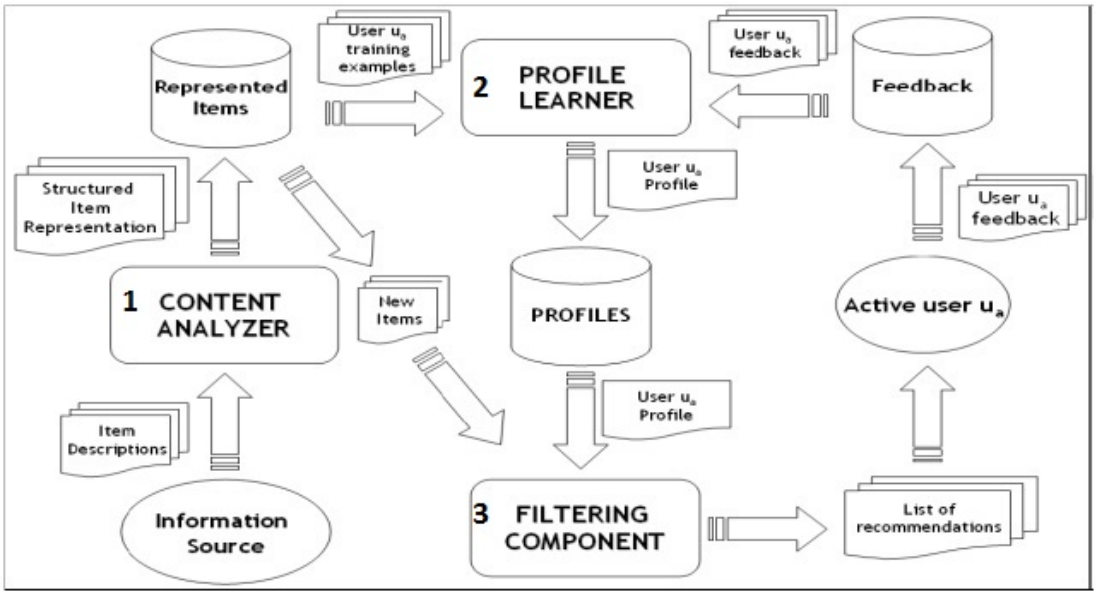


Figure 2.1: Components of content-based recommendation system

In order to construct and update the profile of an active user u_a (user for which recommendations are provided), reactions to items are collected in some way and recorded in the repository "Feedback". These reactions, called annotation-Kendall[1] or feedback, together with the related item descriptions, are exploited during the process of learning a model useful to predict the actual relevance of newly presented items. Users can also explicitly define their areas of interest as an initial profile without providing any feedback.

2.1.1 Content-based Filtering: Challenges and Advantages

Challenges:

- Limited Content Analysis-Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge

[1]Kendall-1990

is often needed, e.g., for movie recommendations, the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like. Some representations capture only certain aspects of the content, but there are many others that would influence a users experience. For instance, often there is not enough information in the word frequency to model the user interests in jokes or poems, while techniques for affective computing would be most appropriate. Again, for Web pages, feature extraction techniques from text completely ignore aesthetic qualities and additional multimedia information.

- Over-Specialization- Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called serendipity problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty. To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be recommended just that kind of movies. A perfect content-based technique would rarely find anything novel, limiting the range of applications for which it would be useful.

- New User - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations [34].

Advantages:

- User independence - Content-based recommenders exploit solely ratings provided by the active user to build her own profile. Instead, collaborative filtering methods need ratings from other users in order to find the "nearest neighbors" of the active user, i.e., users that have similar tastes

since they rated the same items similarly. Then, only the items that are most liked by the neighbors of the active user will be recommended.

- Transparency - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes liked that item;

- New Item - Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects collaborative recommenders which rely solely on users' preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

- Limited Content Analysis - Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like. Some representations capture only certain aspects of the content, but there are many others that would influence a user's experience. For instance, often there is not enough information in the word frequency to model the user interests in jokes or poems, while techniques for effective computing would be most appropriate. Again, for Web pages, feature extraction techniques from text completely ignore aesthetic qualities and additional multimedia information [34].

2.2 Collaborative Filtering

CF models try to capture the interactions between users and items that produce the different rating values. However, much of the observed rating values are due to effects associated with either users or items, independently of their interaction. A principal example is that typical CF data exhibit large user and item biases - i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. Because these predictors tend to capture much of the observed signal, it is vital to model them accurately. Such modeling enables isolating the part of the signal that truly represents user-item interaction, and subjecting it to more appropriate user preference models denoted by , the overall average rating. A baseline prediction for an unknown rating $r_{u,i}$ is denoted by $b_{u,i}$ and accounts for the user and item effects [34].

$$B_{ui} = \mu + b_u + b_j$$

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. For example, suppose that we want a baseline predictor for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, , is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline predictor for Titanics rating by Joe would be 3.9 stars by calculating $3.7 - 0.3 + 0.5$. In order to estimate b_u and b_i one can solve the least squares problem

Among collaborative filtering approaches the most popular one is the neighborhood based approach. They do so because of their simplicity, efficiency and their ability to produce personalized and accurate predictions [34].

Unlike content based filtering, collaborative tends to keep the target user in the limelight and thus aims to find users similar to a target user. The main idea is that if two users u and v have similar interests then they will tend to rate or buy products in a similar pattern. Exploiting this key idea, collaborative filtering first finds such similar user; thus forming a neighborhood of similar

users which is later analyzed to find what items to recommends to the target user.

Collaborative filtering approach can be classified into two classes namely *memory* and *model* based approach.In the memory based approach, the user-item rating information available already is used to predict ratings for unrated items.This can be done either by *user-based* or *item-based* method.The user based method approaches the task with information like User u rated the product ,lets call this $R(u)$.It then finds neighbors of User u(users who have similar tastes) and thus predicts rating for an item using this information.

2.2.1 Collaborative Filtering: Challenges and Advantages

Challenges:
There are many challenges for collaborative filtering tasks. CF algorithms are required to have the ability to deal with very sparse data, to scale with the increasing numbers of users and items, to make satisfactory recommendations in a short time period, and to deal with other problems like synonymy (the tendency of the same or similar items to have different names), shilling attacks, data noise, and privacy protection problems. E-commerce recommendation algorithms often operate in a challenging environment, especially for large online shopping companies like eBay and Amazon. Usually, a recommender system providing fast and accurate recommendations will attract the interest of customers and bring benefits to companies. For CF systems, producing high-quality predictions or recommendations depends on how well they address the challenges, which are characteristics of CF tasks as well.

- Data Sparcity: Due to large product number of products and users the user-item matrix is often very sparse and thus the predictions or recommendations are not very effective. One of the most common problem in which data sparsity is a major challenge is the *Cold-start* problem. Cold-start problem occurs when a new user or item just enters the system. An item does not appear in the recommendations until some users provide ratings for that item and users are not provided recommendations because of less information in their purchase history.

- Scalability: When numbers of existing users and items grow tremendously, traditional CF algorithms will suffer serious scalability problems, with computational resources going beyond practical or acceptable levels. For example, with tens of millions of customers(M) and millions of distinct catalog items (N), a CF algorithm with the complexity of $\bigcirc(MN)$ is already too large. Also, many systems need to react immediately to online requirements and make recommendations for all users regardless of their purchases and ratings history, which demands a high scalability of a CF system [26][34].

Advantages:

- Simplicity- Neighborhood-based methods are intuitive and relatively simple to implement. In their simplest form, only one parameter (the number of neighbors used in the prediction) requires tuning.

- Justifiability- Such methods also provide a concise and intuitive justification for the computed predictions. For example, in item-based recommendation, the list of neighboring items, as well as the ratings given by the user to these items, can be presented to the user as a justification for the recommendation. This can help the user easily understand the recommendation and its relevance, and could serve as the basis for an interactive system where users can select the neighbors for which a greater importance should be given in the recommendation [4].

- Efficiency- One of the strong points of neighborhood-based systems is their efficiency. Unlike most model-based systems, they require no costly training phases, which need to be carried out at frequent intervals in large commercial applications. While the recommendation phase is usually more expensive than for model-based methods, the nearest-neighbors can be pre-computed in an offline step, providing near instantaneous recommendations. Moreover, storing these nearest neighbors requires very little memory, making such approaches scalable to applications having millions of users and items.

- Stability- Another useful property of collaborative filtering approach is that they are little affected by the constant addition of users, items and

ratings, which are typically observed in large commercial applications. For instance, once item similarities have been computed, an item-based system can readily make recommendations to new users, without having to re-train the system [34].

2.3 Neighborhood Based Recommendations

Collaborative filtering can be represented as the problem of predicting missing values in a user-item ratings matrix.Figure 2.2 shows a simplified example of a user-item ratings matrix. For example, in the figure, User 2 assigns rating of "2" to Item 1, whereas the active user a assigns rating of "5" to Item 2. Here, the active user a is the user whose ratings are to be predicted.

In the neighborhood-based algorithm [16], a subset of users is first chosen based on their similarity to the active user, and a weighted combination of their rating is then used to produce predictions for the active user. This algorithm is summarized in three steps:

1. Weigh all users with respect to similarity to the active user. This similarity between users is measured as the Pearson correlation coefficient between their rating vectors.

2. Select n users that have the highest similarity with the active user. These users form the *neighborhood*.

3. Compute a prediction from a weighted combination of the neighbor's ratings.

In Step 1, $S_{a,u}$, which denotes similarity between users a and u, is computed using the Pearson correlation coefficient defined below:

$$S_{a,u} = \frac{\sum_{i=1}^{I}(r_{a,i} - \bar{r}_a) \times (r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i=1}^{I}(r_{a,i} - \bar{r}_a)^2 \times \sum_{i=1}^{I}(r_{u,i} - \bar{r}_u)^2}}, \tag{2.3.1}$$

where $r_{a,i}$ is the rating given to Item i by User a, and $\bar{r}_a$ is the mean rating given by User a, and I is the total number of items.

In Step 2, a subset of appropriate users is chosen based on their similarity to the active user, and a weighted aggregate of their ratings is used to generate predictions for the active user in the next Step 3.

In Step 3, predictions are computed as the weighted average of deviations from the neighbor's mean:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u=1}^{n}(r_{u,i} - \bar{r}_u) \times S_{a,u}}{\sum_{u=1}^{n} S_{a,u}}, \tag{2.3.2}$$

where $p_{a,i}$ is the prediction for active user a for item i. $S_{a,u}$ is the similarity between users a and u as described in Equation (2.3.1). n is the number of users in the neighborhood.

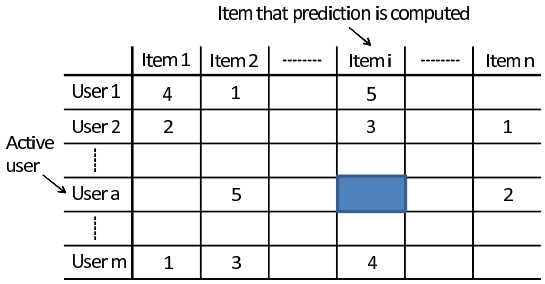


Figure 2.2: User-item ratings matrix for collaborative filtering.

2.4 Graph Based Recommendations

In graph based approach data is represented in the form of a graph where the nodes can represent users or items or both and the edges represent the interactions or similarity between them. Graphs can inherently represent information which is unexploited by traditional collaborative filtering methods. The transitive relations between nodes can be used to derive recommendations that might be of interest to a user maybe not directly but transitively.

For example, in Figure 2.3,nodes represent both users and items. The edges represent the interaction between the user and item. The interaction can be of various types. Presence of an edge could represent that the item was bought

by the user. The edge could even carry some weight which could indicate the rating of that user for that particular item. These weights could then be used to calculate the proximity of a user and item.

In Figure 2.4, few possible ways of using graphs to represent user-item, user-user, item-item interactions are shown. In Figure 2.4a an item's rating is predicted by using ratings of items that are connected to that item. This is the standard approach to follow for predicting rating of unrated item. But this phenomenon can be leveraged by allowing edges to carry information that is more than merely an interaction.It can provide a way to traverse items that are not directly connected but still related to a particular item. In Figure 2.4b the rating for an item is predicted by calculating proximity between a user and the item on the graph. Another way to approach is shown in Figure 2.4c where nodes simply represent users and the edges are the similarity between those users.This approach could alternatively be used for items as well.

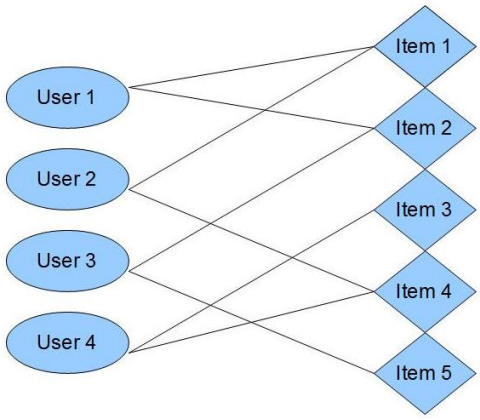
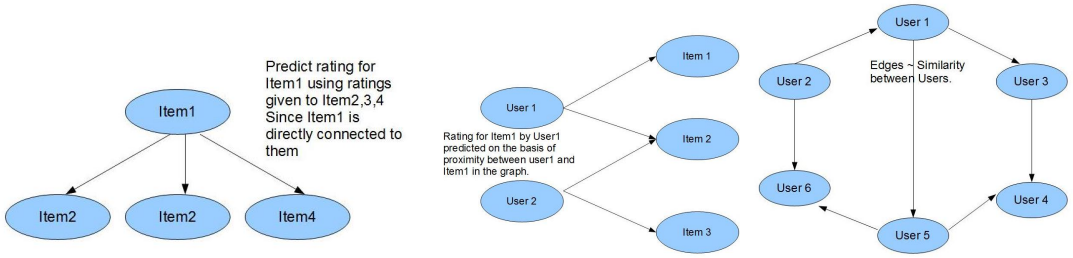


Figure 2.3: Graph representing user-item interactions.

Similarity measures in graph based approach are as follows:

- Path based Similarity- The distance between two nodes of the graph is evaluated as a function of the number of paths connecting the two nodes as well as the length of these paths [34].
- Random walk similarity- Similarity between users or items is evaluated as probability of reaching these nodes in a random walk.This process is



(a) (b) (c)

Figure 2.4: Various graph-based approaches for representing users, items or both.

often described in the form of a weighted graph having a node for each state and where the probability of jumping from one node to another is given by the weight of the edge connecting these nodes [34].

Graph-based techniques exploit the transitive relations in the data. These techniques also avoid the problems of sparsity and limited coverage by evaluating the relationship between users or items that are not directly connected. However, unlike dimensionality reduction, graph based methods also preserve some of the local relations in the data, which are useful in making serendipitous recommendations.

2.4.1 Summary

This chapter discussed the state-of-the-art recommender systems. It provided a detailed study of types of recommender systems followed by their individual challenges and advantages. Also it introduced graph based approach.The next chapter presents the literature review pertaining to the proposed approach.

Chapter 3

Literature Review

Chapter 2 provided an overview of state-of-the-art in Recommendation systems. Also it pointed out the problems with existing approaches. This chapter introduces serendipitous recommendation. Section 3.1 explains the issues with existing approaches for RSs. Section 3.2 elaborates on why accuracy is not the only factor to be concerned about when dealing with recommendations these days. The key focus of section 3.3 and 3.4 is to understand the concept of serendipitous recommendations and the existing work in this domain. Finally, Section 3.6 provides a summary of this chapter.

3.1 Basic Approach to Recommendations

In order to achieve serendipitous recommendation we need to review some problems of RSs which need to be solved. They are as follows:

- Cold start problem: Schein[31] developed a method for recommending items that combine content and collaborative data under a single probabilistic framework. Recommender systems have to deal with the cold start problem as new users and/or items are always present. Rating elicitation is a common approach for handling cold start. However, there still lacks a principled model for guiding how to select the most useful ratings.Lin et al.[22] proposed a principled approach to identify representative users and items using representative-based matrix factorization. The method is significantly effective in achieving good balance between coverage and

diversity and demonstrate that ratings on the selected representatives are much more useful for making recommendation. Building up on his previous work Liu et al. [21] recognised the importance of multiple selection criteria to improve the recommendation output. They proposed a principled approach to identify representative users and items using representative-based matrix factorization. The most common way of dealing with cold start is to leverage metadata for estimating taste of new users, but metadata is not always available, so ask user for some ratings on seed items. The author proposes a principled approach to find seed items by locating representatives. Representatives are those users whose linear combination of tastes would accurately approximate other users.

- Data sparsity problem: Zhou et al. [39] presented a functional matrix factorization (FMF), a novel cold-start recommendation method that solves the problem of initial interview construction within the context of learning user and item profiles.

  Moshfeghi et al. [28] provides a frame- work which is able to tackle sparsity issues by considering item-related emotions and semantic data. In order to predict the rating of an item for a given user, this framework relies on an extension of Latent Dirichlet Allocation [7] , and on gradient boosted trees for the final prediction.

- Search Result Diversification: Ziegler et al. [40] proposed the idea of moving beyond pure accuracy and towards overall user experience. He introduced topic diversification emphasizing that both accuracy and extent of user's interest affect of TD on both user-based and item-based CF algorithms. Moving over regular accuracy metrics like precision and recall he introduced Intra List Similarity.Lower the ILS, higher is the diversity of recommendations.

Topic Diversification: Basically we take a product from the candidate set and compare it with all the items preceding it in rank and then sort all the products based on the similarity calculated in the reverse order obtain- ing dissimilarity rank.

Now merge this rank with the original recommendation rank(according to a factor that determines the impact that dissimilarity rank exerts on overall

output). Experiment conducted confirmed the impact of TD on both user based and item based CF.

Though diversification appeared detrimental to both item and user based CF along precision and recall metrics. Online and offline experiments were conducted: Online experiments of ILS suggests that effect of TD on item based CF is much stronger than user based CF. Limitation of this approach was in respect with the taxonomies(the problem of what human perceive as diverse and what the algorithm does). The primary premise upon which top-*N* recommender systems operate is that similar users are likely to have similar tastes with regard to their product choices.

For this reason, recommender algorithms depend deeply on similarity metrics to build the recommendation lists for end-users. However, it has been noted that the products offered on recommendation lists are often too similar to each other and attention has been paid towards the goal of improving diversity to avoid monotonous recommendations. Noting that the retrieval of a set of items matching a user query is a common problem across many applications of information retrieval, we model the competing goals of maximizing the diversity of the retrieved list while maintaining adequate similarity to the user query as a binary optimization problem.

Zhang et.al [38] explore a solution strategy to this optimization problem by relaxing it to a trust-region problem. This leads to a parameterized eigen value problem whose solution is finally quantized to the required binary solution. They apply this approach to the top-*N* prediction problem, evaluate the system performance on the Movielens dataset and compare it with a standard item-based top-*N* algorithm. A new evaluation metric Item Novelty is proposed in this work. Improvements on both diversity and accuracy are obtained compared to the benchmark algorithm. Collaborative Filtering (CF) algorithms, used to build web based recommender systems, are often evaluated in terms of how accurately they predict user ratings. However, current evaluation techniques disregard the fact that users continue to rate items over time: the temporal characteristics of the system's top-N recommendations are not investigated. In particular, there is no means of measuring the extent that the same items are being recommended to users over and over again. Lathia et al. [20] shows that temporal diversity is an important facet of recommender

systems, by showing how CF data changes over time and performing a user survey. They then evaluate three CF algorithms from the point of view of the diversity in the sequence of recommendation lists they produce over time. They examine how a number of characteristics of user rating patterns including profile size and time between rating affect diversity. They propose and evaluate set methods that maximise temporal recommendation diversity without extensively penalising accuracy [38]. Ziegler's work was for increasing user satisfaction, but the downside was that accuracy was compromised. Though they presented greater number of classes as compared to traditional techniques but failed to analyze or measure item novelty. The novelty is measured in terms of distance from the present interest to the item recommendation in the taxonomy. This approach does this using random walk with restarts [29]. Boim et.al [8] considers a popular class of recommender systems that are based on Collaborative Filtering (CF) and proposes a novel technique for diversifying the recommendations that they give to users by clustering the items based on a unique notion of prioritymedoids that provides a natural balance between the need to present highly ranked items vs. highly diverse ones.

3.2 "Accuracy Does Not Tell the Whole Story": Need for Serendipity in Recommendations

Accuracy is the probability that the active user will appreciate the products recommended. Existing recommendation technologies try to get the recommendation lists with accuracy as high as possible. Their aim is to maximize precision/recall but they neglect aspects of recommendation like novelty and serendipity. *novelty* and *serendipity* are metrics to measure the non-obviousness of recommendations. If you go to Amazon, recommendations will appear in similar purchased or you may also like section. They often tend to be very restricted in terms of the diversity of items that are recommended. Most of the times the recommendations made are far too obvious and offer little variety. For a user, it might not be the best things because they want "good recommendations" and not ones based purely on accuracy. So we have to move beyond accuracy.Recommendation lists are now considered as entities in their

own rights.

Content-based systems suffer from over-specialization, since they recommend only items similar to those already rated by users. One possible solution to address this problem is the introduction of some randomness. For example, the use of genetic algorithms has been proposed in the context of information filtering [1]. In addition, the problem with over-specialization is not only that the content-based systems cannot recommend items that are different from anything the user has seen before. In certain cases, items should not be recommended if they are too similar to something the user has already seen, such as a different news article describing the same event. Therefore, some content-based recommender systems, such as Daily-Learner [6], filter out items if they are too similar to something the user has seen before. Zhang et al. [38] proposed the use of redundancy measures to evaluate whether a document that is deemed to be relevant contains some novel information as well. In summary, the diversity of recommendations is often a desirable feature in recommender systems.

Serendipity in a recommender can be seen as the experience of receiving an unexpected and fortuitous item recommendation, therefore it is a way to diversify recommendations. While people rely on exploration and luck to find new items that they did not know they wanted (e.g., a person may not know she likes watching talk shows until she accidentally turns to David Letterman), due to over-specialization, content-based systems have no inherent method for generating serendipitous recommendations, according to Gups theory [15].

It is useful to make a clear distinction between novelty and serendipity. Herlocker [16] explained that novelty occurs when the system suggests to the user an unknown item that she might have autonomously discovered. A serendipitous recommendation helps the user to find a surprisingly interesting item that she might not have otherwise discovered (or it would have been really hard to discover).

The injection of serendipity in recommendation lists are often seen as programmatically allowing serendipitous recommendations to make the cut in the process of generating recommendations. From this perspective, the problem has not been deeply studied, and there are really few theoretical and experimental studies.

Toms [35] explains that , there are three kinds of information searching: 1. seeking information about a well-defined object; 2. seeking information about an object that cannot be fully described, but that will be recognized at first sight; 3. acquiring information in an accidental, incidental, or serendipitous manner.

It is easy to realize that serendipitous happenings are quite useless for the first two ways of acquisition, but are extremely important for the third kind. As our discussion concerns the implementation of a serendipity-inducing strategy for a content-based recommender, the appropriate metaphor in a real-world situation could be one of a person going for shopping or visiting a museum who, while walking around seeking nothing in particular, would find something completely new that she has never expected to find, that is definitely interesting for her.

In conclusion, the adoption of strategies for realizing operational serendipity is an effective way to extend the capabilities of content-based recommender systems in order to mitigate the over-specialization problem, by providing the user with surprising suggestions.

3.3 Towards Serendipitous Recommendations

This is a summary of recent work related to serendipitious recommendations in recommender system. Serendipity simply means a ”happy encounter” wherein you stumble upon something you were not explicitly looking for. What role serendipity has come to play in terms of modern web search, how personalization in searches are limiting the scope of serendipitous recommendations, how to achieve diverse recommendation lists, what are the state-of-the-art evaluation metrics for serendipitous recommendation, through this section we will try to study these issues.

Not a lot of work has yet been done in the domain of serendipitous recommendations but slowly people are waking up to the idea of not just relying on accuracy but the ”overall user experience” to be the driving force behind generating recommendation lists. In one of the earliest work, Ziegler et al. [40] recognized this need. Although not strictly in the domain of serendipitous rec-

ommendations, it is an obvious motivation for this thesis. The work presented topic diversification, a novel method designed to balance and diversify personalized recommendation lists in order to reflect the users complete spectrum of interests.

Though being detrimental to average accuracy, they showed that their method improves user satisfaction with recommendation lists; in particular for lists generated using the common item-based collaborative filtering algorithm. They also introduced the intra-list similarity metric to assess the topical diversity of recommendation lists and the topic diversification approach for decreasing the intra-list similarity.

Relevance of Ziegler's work in todays scenario with the sudden surge of smart phones and apps- The essence of this highly cited paper in the domain of serendipitous recommendations is that there is a need to move beyond the obvious and the introduction of an evaluation metric to evaluate the obviousness. The obvious here being the explicit interest shown by a customer towards a particular category of products which leads to recommendations specific to that category. Everyone appreciates a little change every now and then. This concept is very subjective in nature owing to the fact that you can never be certain if the surprise recommendation was actually appreciated by the end user or not. Well thats something that can be evaluated easily. The difficult part is to identify what to recommend. More urgent than this is addressing the problem of increased cardinality of the domain we are dealing with. As the number of products increase the consumption rate is pushed in the opposite direction. The consumers are not able to experience the products at a rate proportional to the rate at which these products are being pushed into the market.

The domain of conventional products like movies, books, CDs wherein the number of products being introduced in the market is very moderate, current recommendation engines work sufficiently good but in domains like mobile applications where a substantial amount of applications are introduced every week these same recommendation strategies fail to recommend novel items to the end user. Please refer Figure 3.1.

Figure 3.1: Pie chart depicting the share of books, movies, mobile apps in the market

If we compare the domains of Books, Movies and Apps, the most distinct is the number of products in a domain and the actual number of products that are used by the end user. The cardinality is maximum in the domain of mobile applications with over 4.5 million applications. Also there is a limited set of products that are actually experienced by the end user.Please refer Figure 3.2 to see the difference between available products and products actually experienced. Making recommendation for such products is challenging since nobody has ever experienced these products.

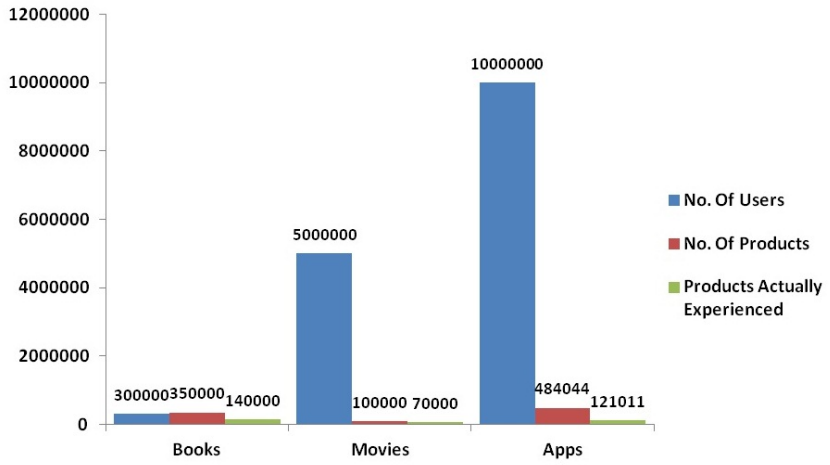


Figure 3.2: Bar graph with estimates of how much percentage of products are actually experienced by the end user

With the current market scenario it can be said that there is a need for more advanced and efficient recommendation techniques that can move beyond the obvious recommendations and suggest products that have not been experienced yet but potentially novel to the end user.Ziegler et al. [40] had used few condensation steps to bring the dataset from 1,157,112 ratings to 361,349 ratings in order to get more accurate findings from the collaborative filtering algorithm method used. This is a very common practice for doing research in collaborative filtering approach. But for domains with massive cardinality the condensation steps followed by the author can push us further and further away from reality. It is to be believed that over 2/3rd of the products are not ranked, using steps like removing products with less than 20 mentions will always discard these infrequently experienced items. But a very few percentage of users are in fact experiencing them so these transactions are important.

Neighborhood formation is at the very heart of recommender systems. By finding out users similar to our target user we construct the user-user similarity matrix and then make the predictions/recommendations. Both with user-based and item-based collaborative filtering algorithm, some similarity measure is used to compare two users or two items. The approach used by the author is to use a taxonomy to classify all the items and use average of all the ratings for two products in order to compare them. Now by simple law of average its easy to prove that too many lower rankings will pull the fewer high rankings to give an average ranking which is not close to the real time scenario. Considering work done strictly in the domain of serendipitous recommendations, Andre et al. [4] came up with some significant observations. Highly accurate search engines reduce the opportunities for serendipity. Personalization is another deterrent for serendipity by giving user exactly what he needs. There are two important aspects to understand serendipity, firstly what activity was user engaged in at the time of the serendipitous encounter and secondly what type of information was encountered. Whether the user was doing a goal directed browsing/non directed browsing, relevant/irrelevant information. There's value in partially relevant searches since they are interesting but not directly related to the user's interest, but they are potential candidates for serendipitous recommendations. A Low click entropy would mean a small

number of results for clicked for a particular query. High click entropy would mean a high number of different results were clicked for a particular query. Aim is to move towards positive correlation between click entropy and results that were judged interesting or potentially serendipitious. Kawamae [18] work on serendipity via innovators is an approach of selecting representatives for each user. This helps in solving the cold-start problem very effectively. Sugiyama and Kan [33] proposed a framework for scholarly paper recommendations.User profile construction is done using information derived from

- dissimilar users and
- co-authors to specifically target serendipitous recommendation.The proposed method has three main steps.

Firstly, the user profile is constructed for every researcher. Using this profile,another profile is constructed but this time concentrating on the serendipitous recommendations. Feature vectors are then constructed to recommend candidate papers using citations of the target paper.Finally using cosine similarity, recommendations are generated.

3.4 Evaluation Metrics for Serendipitous Recommendations

Evaluation measures act as objective functions to be optimized by information retrieval systems. Such objective functions must accurately reflect user requirements, particularly when tuning IR systems and learning ranking functions. Ambiguity in queries and redundancy in retrieved documents are poorly reflected by current evaluation measures. In regards to this Clarke et.al [11] introduces a framework for evaluation that systematically rewards novelty and diversity. This framework was developed into a specific evaluation measure, based on cumulative gain demonstrate the feasibility of the approach using a test collection based on the TREC question answering track.

Kawamae [18] came up with the idea of utilizing the surprise of each user in the recommendation process by focusing on the estimated search time that the users would take to find the item by themselves. He assumed that items

recently purchased by an ”innovator” will surprise other users more than other items.It is like assigning a user with unpredictable traits as a representative for other users.It also takes into account the fact that user interests change over a passage of time and also the trends of products keep changing.The reason for doing this is quite understandable since the existing metrics do not account for the dynamics that user preferences goes through.

Nakatsuji [29] proposed a method for identifying items that are highly novel for the user by identifying the smallest distance from the class accessed before the class where the actual target item is. Dissimilarity metrics are also being researched extensively. In this domain Aktolga [2] came up with dissimilar measures to detect outlier sections in congressional legislation so that readers can easily understand long legislation documents. The dissimilarity measures employed are KL divergence, JS divergence, KL divergence contribution, symmetric KL divergence contribution. Ziegler et al. [40] came up with new metric like ”Intra List Similarity” for Topic Diversification. Nakatsuji [29] further improved on the shortcomings of Ziegler et al. by proposing a method for identifying items that are highly novel for the user, by defining item novelty as the smallest distance from the class the user accessed before the class that actually contains the target item. Regarding serendipity evaluation, there is a level of emotional response associated with serendipity that is difficult to capture, therefore an effective serendipity measurement should move beyond the conventional accuracy metrics and their associated experimental methodologies. New user-centric directions for evaluating new emerging aspects in recommender systems, such as serendipity of recommendations, are required [25].

Another work by Celma and Herrera [10] presents two methods, Named Item and User Centric, to evaluate the quality of novel recommendations. The former method focuses on analysing the item based recommendation network. The aim is to detect whether the network topology has any pathology that hinders novel recommendations. The latter, user centric evaluation, aims at measuring users perceived quality of novel recommendations. The results of the experiments, done in the music recommendation context, show that last.fm social recommender, based on collaborative filtering, is prone to popularity bias. This has direct consequences on the topology of the item based recom-

mendation network. Pure audio content based methods (CB) are not affected by popularity. However, a user centric experiment done with 288 subjects shows that even though a social based approach recommends less novel items than our CB, users perceived quality is better than those recommended by a pure CB method.

The Recommender Systems community is paying increasing attention to novelty and diversity as key qualities beyond accuracy in real recommendation scenarios. Despite the raise of interest and work on the topic in recent years, we find that a clear common methodological and conceptual ground for the evaluation of these dimensions is still to be consolidated. Different evaluation metrics have been reported in the literature but the precise relation, distinction or equivalence between them has not been explicitly studied. Furthermore, the metrics reported so far miss important properties such as taking into consideration the ranking of recommended items, or whether items are relevant or not, when assessing the novelty and diversity of recommendations.They present a formal framework for the definition of novelty and diversity metrics that unifies and generalizes several state of the art metrics. We identify three essential ground concepts at the roots of novelty and diversity: choice, discovery and relevance, upon which the framework is built. Vargas and Castells [36] introduced item rank and relevance are introduced through a probabilistic recommendation browsing model, building upon the same three basic concepts. Based on the combination of ground elements, and the assumptions of the browsing model, different metrics and variants unfold.

3.5 Summary

This chapter laid the foundation needed to understand the motivation behind the proposed approach.It explained what is the need for serendipity in recommendations and why we need to go beyond the accuracy metrics like precision and recall. It then explained the various aspects and issues in achieving serendipity. The next chapter illustrates the techniques that constitute the proposed approach to address the issues presented in this chapter.

Chapter 4

Proposed Methodology

Section 4.1 introduces the shortcomings of existing approaches. Section 4.2 explains the preliminaries for the proposed system which includes data collection. Section 4.6 illustrates the system architecture of the proposed graph based approach, describes each component of the system and shows how each of the components of the proposed technique contributes to the recommendation process. Finally, Section 4.7 gives the algorithm and details about the various components.Finally section 4.8 gives the summary of the chapter.

4.1 Intuition: Why the Graph Approach is Employed?

Graph-based techniques exploit the transitive relations in the data. These techniques also avoid the problems of sparsity and limited coverage by evaluating the relationship between users or items that are not directly connected. However, unlike dimensionality reduction, graph based methods also preserve some of the local relations in the data, which are useful in making serendipitous recommendations.

Similarity graphs are able to capture the inherent interest pattern of a user who has downloaded a set of applications and not all the applications. The main idea is that on the user preference graph if two apps are connected by a higher weighted edge then these apps are similar. Alternatively, if there exists a path connecting these two nodes and the weights on each edge that constitutes this path are all sufficiently large or statistically speaking the sub graph

density for this path is high then apps along this path which are not already downloaded with a user are good candidates for surprise recommendation for the user.

4.2 Existing Recommendation Techniques: Where They Lack.

Recommendation techniques are known to have their share of problems, most common being the *new user* and *new item* problem.
New User Problem is the problem of predicting/suggesting items to a new user since very little data is available about his preferences. Along with this is the problem of *New item*, which means that if an item has very little ratings, it does not appear in the recommendations for quite some time till sufficient ratings are available.

"Item discovery" becomes one of the major concern for researchers as the rate at which items are being pushed in the market is increasing exponentially thus making the task of item discovery more difficult.

This is also true in the domain of mobile applications(popularly known as "apps"). There is a constant stream of new apps being added to the various app stores. These apps do not get experienced by that many users for them to show up in the recommendations. Thus, even though there is a huge pool of available apps only a subset gets recommended. This is the *early rater* problem. This phenomenon is similar to the *portfolio effect* [3], which suggests a list of very similar items. So suggesting movies of same director, set of actors, genre to a user who has shown preference for them may not be good for an overall satisfying user-experience even though the accuracy of the list might be very high.

As discussed earlier, item-based collaborative systems can suffer from this effect [40]. A personal experience of visiting the Amazon store backed this observation. Browsing novels for Danielle Steel[1] a couple of years back, my recommendations constituted of other novels by the same author. In economics, its known as law of diminishing marginal utility[32]. The law describes effects

[1] http://daniellesteel.com

of saturation that steadily decrease the incremental utility of products p when acquired or consumed over and over again. For example, suppose you are offered your favorite drink. Let $p1$ denote the price you are willing to pay for that product. Assuming you are offered a second glass of that particular drink, the amount $p2$ of money you are inclined to spend will be lower, i.e., $p1 > p2$. Same for $p3, p4$, and so forth.

A very specific aspect of content recommender systems is relevant to this discussion. Due to the nature of this kind of systems, they can only recommend items that score highly against a userś profile, thus the user is limited to being recommended items similar to those already rated. This shortcoming, called over-specialization, prevents these systems to be effectively used in real world scenarios.

The proposed approach tries to address this issue in a way that users get more diverse items recommended without compromising a lot on accuracy.

4.3 Preliminaries

This section describes the details of representing the information content (IC) of mobile applications, computing app-to-app similarity weights and constructing the app-to-app similarity matrix.

4.3.1 Data Collection

Data collection related to mobile apps is a challenging task. There are two major native app stores for mobile market, namely, Apple iTunes for iOS apps and Google Android market for Android apps. To analyze the mobile app market, we require collecting data of mobile apps from both. In both stores, there are the following two types of data for each mobile app:

- Static data, such as App name, description, developer name that does not change over time and
- Dynamic data, such as rating, rank, reviews that changes on a daily basis.

The static data can be collected by one-time crawling, but the dynamic data needs to be collected by crawling the native stores on daily basis. This creates a

challenge. The additional challenge in data collection is the format of the data available in native stores - iTunes and Android market. It is impossible for a standard crawler to retrieve the data from native stores; we need to build a specialized crawler capable of handling AJAX and JSON technologies. Instead of developing the data collection component from the scratch, for this particular work, data was collected by a commercial project - Mobilewalla . Mobilewalla is a venture capital backed company that specializes in collecting, analyzing and presenting data related to mobile apps in four native stores Apple iTunes, Google Android market, Blackberry native store and Windows App store.

Android and Apple iOS form 70% of the mobile app market worldwide. Additionally USA is the largest market for mobile apps 41% of worldwide mobile app market in 2011. Thus for this particular research, our focus was on mobile apps in iTunes stores in USA only. Since this work required natural language processing, data-set included only English language apps, i.e., apps whose title and descriptions are written in English. Additionally, due to realistic limitation in a research environment, data from May 2011 to January 2012 was considered to select the most popular apps in the store.

This work uses two main datasets

- *App Data:* This is the data about apps like their Title, Category, Description and any other descriptive data. This has been collected from Mobilewalla[1].

- *User Data:* This data has been collected by doing a survey with campus students. We asked students to list the names of the apps downloaded on their iphone(thus restricting apps to apple appstore). The dataset thus contains username and name of the app installed. This information is then used to generate the recommendations.

For the sake of notations, we are going to call the datasets as *app Data* and *user data* in all our future references

[1]http://mobilewalla.com

Table 4.1: App Attributes

App ID	Unique ID for every app in a particular store
App Title	Name of the app
App Category	Category the app belongs to i.e lifestyle, entertainment, sports, travel etc
App Description	Description of the application
Price	Free/Paid
Dev ID	Unique ID given to the developer of the app
Seller ID	Unique ID given to the seller of the app
Latest Version	Different versions of an app exist in the app store
Update Date	Date on which the last update was performed
Platform	Whether the App is an Apple, Android, Blackberry or Windows based
Size	Size of the App in Kb/Mb
Language	Language in which the app is available

4.3.2 App Representation

There is a vast amount of information available for mobile applications popularly called mobile apps. Ranging from app characteristics to download statistics to developer details.

Information available about Mobile apps(Mobilewalla) is as follows:
If we talk about information content of a mobile application, only a handful of the textual information can lead us to what the application is actually all about like,what does the app do, what sort of user base it caters to, what the quality of the application is etc. So even though there is a lot of information available about mobile apps, we selected the attributes most relevant to the research problem. Textual Information available in case of mobile apps:

- *Title of the Mobile App*: A title is the basic and most intuitive way of knowing what the App does and thus is an obvious choice.
- *Category of an App*: This indicates what category the apps belongs to and thus helps in finding similar apps.
- *Description*: Most mobile apps have a good description about the functionality of an app, which platform it supports and what are the prime features, it thus helps understand the mobile app better.

These three attributes collectively now represent an *app-document* representing each app. Thus the further processing is done on these app-documents generated by combining these three attributes. The next step is to preprocess these app-documents so as to prepare them for the similarity generation module.

4.3.3 Preprocessing of App data

Most of the time app developer release various versions of their root apps(pro, lite and HD). For example, app called Fruit Ninja has several variants by same developers in both Apple and Android platforms such as Fruit Ninja Pro, Fruit Ninja Lite and Fruit Ninja THD Free. To work with so many apps and their different versions and finally calculate the similarity values between each pair of apps, first we preprocess all the app names and descriptions using the preprocessing algorithm. Usually there are stop words- by, edition, Free, HD, THD, Seasons, lite, pro, Classic, enhanced and symbols like +,!,- which are needed to removed before processing further. The preprocessor algorithm removes these stop words and symbols from the app-documents thus aiding the natural language processing for finding similarities between them. The main steps involved in the preprocessing algorithm are:

- Checking English words- This checks whether the words appearing in the app-documents are English or not. It achieves this by matching the ascii value of letters and checking if they fall within the permissible ascii key set of English alphabets.

- Removing stop words- Stop word removal is a very common step in applications that involve natural language processing. There are list of common stop words available due to a lot of research in this domain. We modified the stop word removal list to suit our application and domain.
- Removing punctuation- After the stop word removal, the algorithm removes punctuation from the app-documents to make them more refined and concentrated.
- Truncation of miscellaneous symbols- This step includes removal of miscellaneous symbols like-(#,$,&...) and some other symbols which do not have a straightforward ascii key value to match and truncate.
- Creating the tokens- Finally after all the preprocessing of the app-documents, tokens are created so that they can be analysed and used for similarity score generation.

Also an issue with the app names is the absence of white spaces between words, for example - Fruit Ninja HD Lite and FruitNinja HD without the space in between Fruit and Ninja. After removal of the stop words, these apps will be named as Fruit Ninja and FruitNinja respectively. To address such scenario, we tokenize these preprocessed app names based on the capital character along with the white space characters, which will tokenize both the app names into Fruit and Ninja.

Once the stop words are removed and app-documents are tokenized, app-app pair similarity are calculated using the ling-pipe APIs (Alias 2008) TF-IDF scheme between app-document pairs [1]. This TF-IDF scheme is based on vector similarity(using the cosine measure of angular similarity) over dampened and discriminatively weighted term frequencies. The basic idea is that two strings are more similar if they contain many of the same tokens with the same relative number of occurrences. We identify all the app-pairs that cross a threshold score value (our case =0.8).

[1] http://lingpipe.com

4.3.4 App-App Similarity

A mobile app vector is now represented by 3 attributes: App Title, App Category and App Description.

To compute the similarity between two apps we use the popular cosine similarity using TF-IDF term weighting to computer similarity between their attributes.Each App can be considered to be represented by a vector in a n-dimension space where each dimension corresponds to a term appearing in the overall vocabulary of words in the given textual information.

Figure 4.1 is a snapshot of the app-app Similarity Scores. As we can see, every app is compared to every other app.

AppID	App1	App2	App3	App4	App5
App1	1	0.2	0.5	0.1	0.3
App2	0.2	1	0.4	0.2	0.3
App3	0.5	0.4	1	0.5	0.2
App4	0.1	0.2	0.5	1	0.2
App5	0.	0.3	0.2	0.2	1

Figure 4.1: App-app similarity matrix

4.3.5 User Preference Representation

Now that the app representation has been discussed, this section will focus on how the user information is incorporated.

For every user, we have a set of apps representing his taste. These set of apps are set of applications already downloaded on the users phone. It is a holistic view of the interest patterns of the user and provides a good foundation for the user preference graph construction.

In this approach we are not taking any explicit input from true user in the form

of reviews or rankings instead we are simply focussing on what he has on his phone to serve as the window to his taste set as far as apps are concerned. This is slightly different from popular recommender systems based on collaborative filtering which rely on ratings and reviews to compare user profiles and item profiles. For example, *User*1 is represented by list apps installed on his phone.So is *User*2 and so on.

*User*1[*AppId*1, *AppId*2, *AppId*3, *AppId*4, *AppId*5...]
*User*2[*AppId*2, *AppId*3, *AppId*4...]
*User*3[*AppId*1, *AppId*2, *AppId*3...]

Figure 4.2 shows a preference matrix that can be constructed using the information about downloaded apps on users phone. For every app that is downloaded the corresponding cell has a value ”1” else ”0”. This is to effectively represent all the apps that the user has on his phone.

UserID	App1	App2	App3	App4	App5
User1	1	1	1	1	1
User2	0	1	1	1	0
User3	1	1	1	0	0
User4	1	1	1	1	1
User5	1	1	0	0	0

Figure 4.2: User preference similarity matrix

4.4 System Architecture and Methodology

After studying the principles and objectives of RSs this work proposes a recommendation system based on item-item similarity graph. The system intends to retrieve serendipitous recommendations from the graph for a particular user. Their are 4 major components:

(1) Similarity Calculation Module.

(2) App-Similarity Graph Construction Module.

(3) User preference Graph Construction Module.

(4) Recommendation Generation Module.

In Figure 4.3 we present the architecture of the proposed approach. The first module generates the similarity scores between every pair of app by using metadata of each app. This information is used by second module which generates an app-app similarity graph. The third module is responsible for generating a user preference graph by finding apps that are already installed in a user's phone and the edges that connect them. The final module generates the recommendations by finding shortest paths amongst the installed apps.

Below is a table to follow the notations used in further sections. Please refer Figure 4.2 for details.

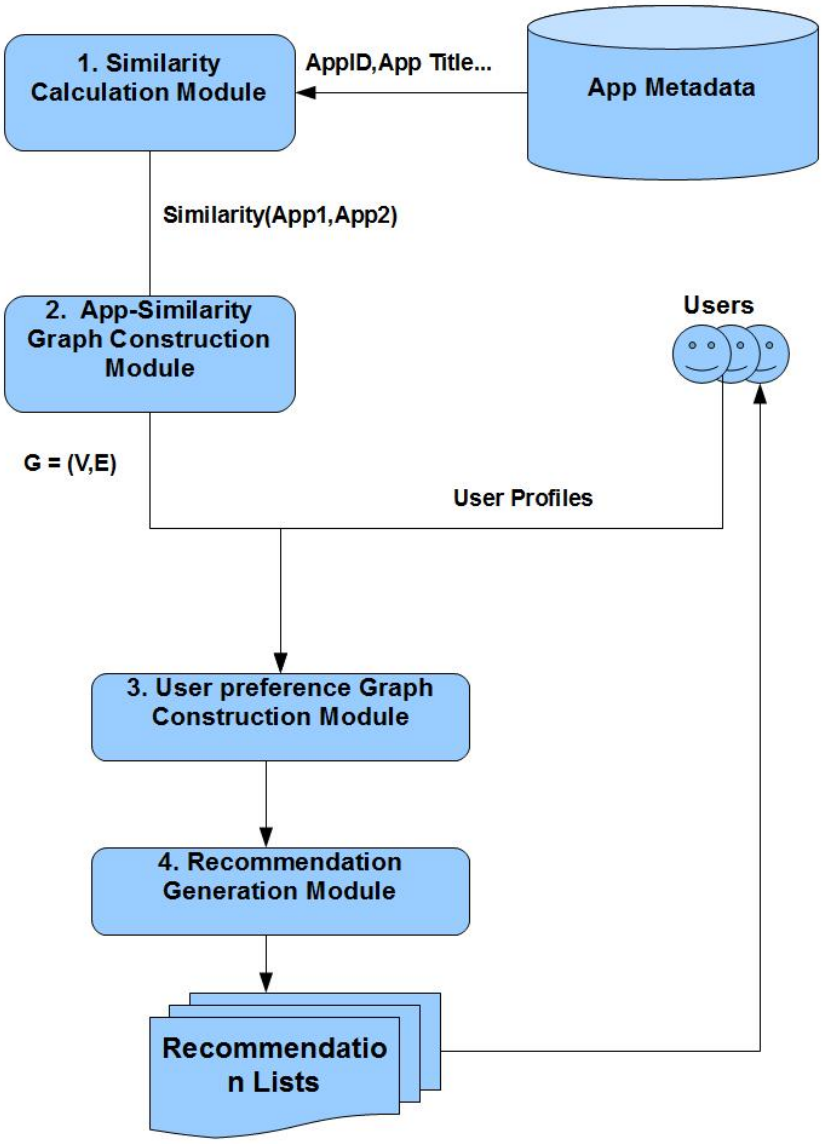


Figure 4.3: System architecture

Table 4.2: Notation

Notation	Meaning
App Data	Data about Apps in the App stores
User Data	Data about Apps used by users
S_A^n	List of App ID's in the App Data
S_{UID}^m	List of App ID's in the User Data
D_A^n	List of App Documents in the App Data
$L_A^{n(n-1)}$	List of App-App Similarity Scores
T_A^n	List of App Titles of Apps in App Data
S_U^m	List of App Titles of Apps in User Data
S_P	List of paths between apps in User Data
E_{A_i,A_j}	List of edges between apps in User Data
G(V,E)	Graph of V vertices and E edges
Simscore(A_i, A_j)	Similarity score between two apps
AddVertex(V_i)	To add a vertex in graph
AddEdge(A_i, A_j)	To add an edge between two vertices
AddWeight(Edge)	To add weight to an edge
TfIdf	To calculate TfIdf score between two texts
n	Total number of Apps in App data
m	Total number of Apps in User data

4.4.1 Similarity Calculation Module

ALGORITHM 1: Similarity Calculation

Input: List of AppID's $S_A^n = \{A_1, A_2...A_n\}$
List of AppDocuments $D_A^n = \{D_1, D_2...D_n\}$
Output: List of App-App Similarity Score
$L_A = \bigcup_{\forall A_i, A_j \in S_A^n, i \neq j} SimScore\{A_i, A_j\}$

```
Size ← |S_A|
L ← ∅
for i = 1; i <= Size − 1; i + + do
    App1 ← A_i
    Doc1 ← D_i
    for j = i + 1; j < Size; j + + do
        App2 ← A_j
        Doc2 ← D_j
        SimScore(A_i, A_j)=tfIdf(Doc_1, Doc_2)
        L_A=L_A ∪ SimScore{App_1, App_2}
    end
end
return L_A
```

Algorithm 1: Similarity Calculation

This algorithm takes input as the list of app IDś and app Documents prepared after the preprocessing stage. This information is then used to construct the graph. The output is a list of similarity scores between every pair of apps.

All the App IDś are considered pairwise. For example, if the list contains three apps, then pairs to consider for similarity calculation are:

(App1,App2),(App1,App3),(App2,App3)

After this step, app documents containing the textual data about each apps are compared using TF-IDF. A function TfIdf.handle() creates token of textual data passed as parameter and adds it to the handle. This is done for both the apps and TfIdf.proximity() calculates the similarity score. This is stored in a separate list which will be later used for graph construction.

4.4.2 App Similarity Graph Construction Module

ALGORITHM 2: App-App Similarity Graph Construction

Input: List of AppID's $S_A^n = \{A_1, A_2...A_n\}$
List of App-App Similarity Scores,
$L_A = \bigcup_{\forall A_i, A_j \in S_A^n, i \neq j} SimScore\{A_i, A_j\}$
Output: Graph $G(V, E)$, $V = \{A_1, A_2...A_n\}$ and $E = \{E_{1,2}, E_{1,3}, E_{2,3}...E_n m\}$
Size $\leftarrow |S_A|$
E $\leftarrow \emptyset$
G $\leftarrow \emptyset$
for $i = 1; i <= Size; i++$ **do**
  $V_i \leftarrow A_i$
  $G.AddVertex(V_i)$
end
for $i = 1; i <= Size - 1; i++$ **do**
  App1 $\leftarrow A_i$
  for $j = i + 1; j < Size; j++$ **do**
    App2 $\leftarrow A_j$
    $G.AddEdge(App_1, App_2)$
    $E\{App1, App2\} = \bigcup \forall A_i, A_j \in S_A^n, i \neq jE\{App1, App2\}$
    $G.AddWeight(SimScore_{App_1, App_2})$
  end
end
return *G*,*E*

This component is responsible for generating the base graph with all the apps in the dataset as vertices and the edges with weight equivalent to the similarity scores. This lays the foundation for generating the user preference graph. Edges contained in this graph are above a certain threshold to allow novel but serendipitous recommendation generated by the subsequent components.

Given the app-app similarity matrix we define a graph *ASG*,

Definition 1: (ASG: App-App Similarity Graph)

Given a set of applications $A = A_1, A_2...A_n$, a graph $G = (V, E)$ is an undirected, fully connected weighted graph where V is node set such that each app

is a node on the graph G. E is edge set, where an edge between V_i and V_j is established if similarity between V_i and V_j is above a certain threshold. Thus, each edge $e \in E$ has an edge weight $0 < W_{ij} < 1$, and $W_{ij} = W_{ji}$.

Algorithm 2: App-App Similarity Graph Construction

This algorithm takes input as list of app IDś and the similarity scores of all the apps available in the dataset. All the app IDś are then added as vertices. For adding the edges the algorithms checks whether there is a similarity score above a certain threshold exists and then creates an edge between those two apps. Also the similarity score is added a weight to that edge. The output is a graph with vertices as the distinct app IDś and edges as similarity values between then. These are typically called Item graphs since the node represents the items and edges between them represent some sort of score between the two.

Post this, all app pairs are considered one at a time and similarity scores are calculated. Also $(1 - similarityscore)$ is stored which is the dissimilarity score which we aim to minimize(in turn maximizing similarity score).

4.4.3 User Preference Graph Construction Module

The user preferences are available in the form of list of app titles that are already installed on a user's phone. These app titles have to be mapped to one of the apps in the dataset in order to work with the similarity scores. The following algorithm compares each user app name with the existing apps in the dataset and assigns the appId where cosine similarity is 1. We thus have a user preference matrix where every app that exists on user's ozone corresponds to a 1 else 0. After the apps have been assigned an appId we can construct the user preference graph.

Given the User Preference matrix we define a graph *UPG*,

Definition 2: (UPG: User Preference Graph)

Given a set of applications $A = A_1, A_2...A_n$ where A_i $(1 < i < N)$ means a downloaded item for a user U, a sub graph of app-app similarity graph comprising of maximum possible applications that a user already on his phone as vertices. This sub graph is generated by finding out the max-weighted edges amongst all vertices and including those edges. This will also include foreign nodes into the graph since maximum weighted paths between two nodes can be through other nodes as well. This helps us get our recommendations.

ALGORITHM 3: Getting User AppID's

Input: List of AppID's $S_A^n = \{A_1, A_2...A_n\}$
List of AppTitles $T_A^n = \{T_1, T_2...T_n\}$
List of UserAppTitles $S_U^m = \{U_1, U_2...U_m\}$
Output: List of UserAppID's $S_{UID}^m = \{UID_1, UID_2...UID_m\}$

```
S^m_UID ← ∅
score ← ∅
for i = 1; i <= m; i++ do
    for j = 1; i <= n; j++ do
        score = tfIdf(U_i, T_j)
        if score = 1 then
            UID_i = A_j
        end
    end
end
return ∅
```

Algorithm 3: Getting User AppID

The user data is in the form of titles of the applications installed on the user phone and thus they need to be resolved into app ID. Now since there are so many application which have highly similar names it would not be easy mapping app Title to its exactly same name. Thus, we wish to find either the exact match or the nearest match for the given app title. For this we calculate the similarity score of every app title with all available app titles and the highest match is found. The app title is thus assigned that particular app id. After this algorithm we have the necessary data to construct the user preference graph which uses the list of app id installed on the user phone as its vertex set and

ALGORITHM 4: User Preference Graph Construction

Input: List of UserAppID's $S^m_{UID} = \{UID_1, UID_2 ... UID_m\}$ List of Edges $E = \bigcup \forall A_i, A_j \in S^n_A, i \neq jE\{App1, App2\}$ Graph $G(V, E)$

Output: Graph $UserGraph(V, E)$, $V = \{UID_1, UID_2 ... UID_m\}$ and $E = \{E_{1,2}, E_{1,3}, E_{2,3} ... E_n m\}$

```
Size ← |S_UID|
G ← ∅
for i = 1; i <= Size; i++ do
    V_i ← UID_i
    UserGraph.AddVertex(V_i)
end
for i = 1; i <= Size − 1; i++ do
    for j = i + 1; j < Size; j++ do
        if ∃E{i, j ∈ G(V, E)} then
            UserGraph.AddEdge(UID_i, UID_j)
        end
    end
end
return UserGraph
```

the connecting edges between them as the edge set.

Algorithm 4: User Preference Graph Construction

In the previous algorithm, we got the list of app id installed on the user phone. Using this data and the app-app similarity graph as our base graph we construct the user preference graph. First of all, the app IDs are added as vertex into the graph. Each pair of App is then checked for an existing edge in the base graph which is the app-app similarity graph.If there exists an edge, then this is included in the user subgraph as well. The output of this algorithm is thus a representation of user's interests on a graph.

4.4.4 Recommendation Generation Module

Algorithm 5: Recommendation Generation

This algorithm takes pairwise apps on the user phone to be considered as

ALGORITHM 5: Recommendation Generation

Input: List of UserAppID's $S^{m}_{UID} = \{UID_1, UID_2 ... UID_m\}$ Graph $G(V, E)$

Output: List of Recommendations R_{user}, List of Paths $S_P^{m*(m-1)} = \{P_{1,2}, P_{1,3} ... P_{m,n}\}$

$Size \leftarrow |S_{UID}|$
$R_{useri} \leftarrow \emptyset$
for $i = 1; i <= Size - 1; i++$ **do**
  for $j = i + 1; j < Size; j++$ **do**
    $Paths \leftarrow FindPaths(UID_i, UID_j)$
    $S_P \leftarrow \bigcup_{\forall UID_i, UID_j \in S^n_A, i \neq j} Paths\{P_i, P_j\}$
  end
end
forall the $Paths \in S_P$ **do**
  $R_{user} \leftarrow CommonVertices(Paths)$
end
return $UserGraph, R_{user}$

source and destination. It then uses Bellman Ford Shortest Path method to calculate shortest path between the source and destination. This is the final step in the proposed method. As mentioned earlier the main idea behind this approach is that on the user preference graph, if two apps are connected by a higher weighted edge then these apps are similar. Alternatively,if there exists a path connecting these two nodes and the weights on each edge that constitutes this path are all sufficiently large then apps along this path which are not already downloaded by a user are good candidates for surprise recommendation for the user.

Banking on this assumption, we construct paths between all the app pairs and create a path list. These paths are then explored to find out the common vertices amongst them other than the source and destination and the ones already downloaded on the user's phone. They are then added to the final recommendation lists and the *UPG* is updated by adding these new recommendations. Since graph is acyclic and connects all of the vertices, it must provide shortest path from every vertex to every other vertex.

By minimizing the dissimilarity function(thereby moving along a path if high similarity) while using Bellman Ford shortest path(which minimizes the

edge weight to find shortest path between source and destination), we are getting a path that connects user downloaded apps with some new apps that are highly similar to these apps and thus are good candidates for serendipitous recommendations.

Thus, the output of this module is a list of serendipitous recommendations for a user on the basis of apps already installed in his phone.

4.5 Summary

This chapter illustrates the proposed system. The system integrates graph based approach to solve to an extent the problem of over-specialization common with content based approaches by providing highly serendipitous recommendations to the usser. The next chapter describes the experimental results of the tests performed to evaluate the techniques presented in this chapter.

Chapter 5

Experiments and Results

5.1 Experiments and Results

This chapter describes the experimental results obtained in the form of an illustration.

5.1.1 Test Data

In this section we first describe the datasets used for generating serendipitous recommendations. The three main datasets used were:

- App Data- This dataset comprises of information about Apps like AppID(across all/local stores), Title of the App, Description of the App etc.
- App-App Similarity Scores- Using the App Data dataset, the similarity scores are generated. This dataset is then further used for constructing the similarity graphs.
- User App Data- This dataset contains the list of App IDs installed on a user's phone. It includes the user name, name of the App installed and the App ID. Initially the data collected were only App Titles but later they were resolved into App ID's for further processing.

Table 5.1: App data about a single app

App Attributes	Data
App ID	127
App Title	”Walt Disney World Guide” Notescast
App Description	”This is a complete guide-book in an app and a very complete one at that.....
Combined Title and Description	Walt Disney World Guide” Notescast This is a complete guidebook in an .....

In 5.1 are presented various attributes of an App that are considered for this work. There is a lot of information about Apps these days but for this work the main focus was on descriptive information rather than statistical information. This attributes like Title and Description have been used. Using the combined title and description in Figure 5.1, we calculate the similarity scores. To do so we user TF/IDF. TF/IDF distance is based on vector similarity (using the cosine measure of angular similarity) over dampened and discriminatively weighted term frequencies. The basic idea is that two strings are more similar if they contain many of the same tokens with the same relative number of occurrences of each. Tokens are weighted more heavily if they occur in few documents. Next section describes the details.

5.1.2 Implementation Details

This work uses LingPipe which an excellent library for various applications of natural language processing. There are two interfaces at the very heart of string comparisons in LingPipe, *util.Distance*$< E >$ and *util.Proximity*$< E >$. Their signatures are both very simple. The distance interface specifies a single method which computes a distance between two objects of type E:

public interface Distance< E > public double distance(E e1, E e2);

The proximity interface also specifies a single method:

public interface Proximity< E > public double proximity(E e1, E e2);

String similarity can be specified in terms of distance. If two strings are more similar, the distance between them will be less. Similarity can also be specified in terms of proximity. If two strings are more similar, the proximity between them will be greater. It's always possible to convert a proximity into a distance or vice-versa by either negation (additive inverse) or inversion (multiplicative inverse). In our case we use the dissimilarity score by calculating (1-similarity) score in order to minimize this while finding paths between pairs of apps. Tokens are generated from the descriptions of the Apps and their document frequency and inverse document frequency is calculated. Below is a simple illustration.

String 1="walt disney world guide notescastcomplete guidebook app complete one app buy firsttuaw com 750 superb photos 300 pages information" String 2="walt disney world guide notescastcomplete guidebook app complete one app buy firsttuaw com 750 superb photos 300 pages information"

Distance=0.00 Proximity=1.00 (This indicates the strings are exactly similar).

String 1="walt disney world guide notescastcomplete guidebook app complete one app buy firsttuaw com 750 superb photos 300 pages " String 2="walt disney world secrets gold notescast now enjoy 250 magical walt disney world secrets 200 fun facts 200 original"

Distance=0.05 Proximity=0.95 (This indicates the strings are almost similar).
These values are calculated for all the app-app pairs. Using this information, app-app similarity graph is constructed. The next step is to provide the user

Table 5.2: Sample user data showing username and apps installed on user's phone

UName	AppName
User1	1001 Ringtones Lite
User1	"10,500+ Cool Facts"
User2	97.9 The Box
User2	2011 World Factbook
User3	97.9 The Box
User3	2011 World Factbook
User4	97.9 The Box
User4	2011 World Factbook
User5	97.9 The Box
User5	100 Ways to Motivate Others by Steve Chandler

dataset which comprises of apps installed on user's phone and the recommendations are generated. The next section presents the recommendations for a *UserA*.

All modules were implemented in Java 1.7 and used MySQL v5.1 to store the similarity scores and recommendation paths. All modules and the database reside in the same computer (a quad processor machine equipped with a 2.33 GHz CPU and 8 GB RAM, and running the Windows operating system).

5.1.3 Results

Using the data about similarity scores between Apps, the App-App Similarity graph is constructed. App ID's are added as vertices and edges are added with weight equivalent to the similarity score between them. Its not possible to show the constructed graph with so many Apps so the recommendations are presented directly.

After the graphs are constructed. Paths are calculated between each pair of user app by minimizing the dissimilarity. In Figure, 5.3, we can see list of apps installed by *UserA* along with their App ID's. Using this data, recommendation are generated. In Figure 5.4, serendipitous recommendations generated for *UserA* can be seen.

The recommendation list includes obvious app recommendations like *"At-*

Table 5.3: Apps installed by user A

AppID	**App Title**
602510	97.9 The Box
3094	2011 World Factbook
1090360	100 Ways to Motivate Others by Steve Chandler
1721	10BII Calc Financial Calculator
4990	5-0 Radio Police Scanner Lite (Free)
1149536	80,000+ Wallpapers HD Free

Table 5.4: Serendipitous recommendations for given user apps

User Apps	**Recommendations**
97.9 The Box	Lock Screen App
2011 World Factbook	Awesome Note(To Do Diary)
100 Ways to Motivate Others by Steve Chandler	High Definition Wallpapers
10BII Calc Financial Calculator	Atlas 2012 Pro
5-0 Radio Police Scanner Lite (Free)	US Weather Maps
80,000+ Wallpapers HD Free	Amazing Car Wallpapers
	Google Apps Browser
	ZipList of Top 100 iPhone Apps
	Office HD2

las 2012 Pro" since user had an installed app, *"2011 World Factbook"*. Other recommendations like *"High Definition Wallpapers"* and *"Amazing Car Wallpapers"* can also be justified. The serendipitous recommendations are

- *"Awesome Note(To Do Diary)"*, which maybe justified since the user uses an app called *"Financial Calculator"*. This may also be understood like this, the user likes daily use based utility apps.
- *"Lock Screen App"*, this is a good serendipitous recommendation for *UserA* since it is an essential for iphone users, but the user has not explicitly shown interest in it.

Chapter 6

Conclusion and Future Work

6.1 Conclusions and Future Work

This work provides a novel way to generate serendipitous recommendations by using information about installed Apps on a user's device. It is independent of ratings/reviews information thus eliminating to an extent the problem of "over specialization" that accompanies content based recommendation systems. Graph based approaches have not been used a lot in serendipitous recommendations thus this work is rather first in this domain. The analysis of the recommendation lists with a larger user dataset needs to be conducted in order to get more conclusive results.

6.1.1 Evaluating serendipitous Recommendations

Novelty in recommendations is often considered to be items that the user did not know about previously and thus are a much welcome break from the list of obvious recommendations. In many applications suggesting similar items is not desirable since it takes longer time to find something useful for a particular user. Thus there are evaluation measures for measuring novelty in recommendations as well.

Very close to this concept is the concept of serendipitous recommendations.The idea of making 'surprise' recommendations which are of likely interest to a user is hard to evaluate. Also evaluating serendipity by comparing with

accuracy would not be a good. Serendipity is a measure of how surprising the successful recommendations are. For example, if the user has rated positively many movies where a certain star actor appears, recommending the new movie of that actor may be novel, because the user may not know of it, but is hardly surprising. Of course, random recommendations may be very surprising, and we therefore need to balance serendipity with accuracy.

The most explored method for measuring diversity/serendipity uses item-item similarity, typically based on item content. Thus the diversity/serendipity of items recommended can be calculated by considering the range of items recommended. The item-item similarity measurement used in evaluation can be different from the similarity measurement used by the algorithm that computes the recommendation lists.

This work can be further extended to find out the extent of serendipity in the recommendation lists. Also more data about users need to be collected and experiments need to be conducted to get more conclusive results. This work thus provides a novel way to generate serendipitous recommendations by using information about installed Apps and is independent of rating/reviews thus eliminating to an extent the shortcoming of ”over specialization” usually with content based recommendation systems.

Bibliography

[1] *Evolving agents for personalized information filtering*, 1993. 37

[2] E. Aktolga, I. Ros, and Y. Assogba. Detecting outlier sections in us congressional legislation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 235–244, 2011. 43

[3] K. Ali and W. van Stam. Tivo: making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 394–401, 2004. 46

[4] P. André, J. Teevan, and S. T. Dumais. From x-rays to silly putty via uranus: serendipity and its role in web search. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 2033–2036, 2009. 41

[5] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: using social and content-based information in recommendation. pages 714–720. American Association for Artificial Intelligence, 1998. 13

[6] D. Billsus and M. J. Pazzani. A hybrid user model for news story classification. In *Proceedings of the seventh international conference on User modeling*, pages 99–108, 1999. 37

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. pages 993–1022, 2003. 34

[8] R. Boim, T. Milo, and S. Novgorodov. Diversification and refinement in collaborative filtering recommender. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 739–744, 2011. 36

[9] R. Burke. The adaptive web. chapter Hybrid web recommender systems, pages 377–408. 2007. 9

[10] O. Celma and P. Herrera. A new approach to evaluating novel recommendations. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 179–186, 2008. 43

[11] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 659–666, 2008. 42

[12] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. on Knowl. and Data Eng.*, 19(3):355–369, 2007. 18

[13] E. Frias-martinez, S. Y. Chen, and X. Liu. Evaluation of a personalized digital library based on cognitive styles: Adaptivity vs. adaptability. 9

[14] E. Frias-Martinez, G. Magoulas, S. Chen, and R. Macredie. Automated user modeling for personalized digital libraries. *International Journal of Information Management*, 26:234–248, 2006. 9

[15] Gup. Technology and the end of serendipity. pages 48–50, 1998. 37

[16] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999. 29, 37

[17] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, 2004. 13

[18] N. Kawamae. Serendipitous recommendations via innovators. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 218–225, 2010. 17, 42

[19] J. K. Kim, H. K. Kim, H. Y. Oh, and Y. U. Ryu. A group recommendation system for online communities. *International Journal of Information Management*, 30:212–219, 2010. 9

[20] N. Lathia, S. Hailes, L. Capra, and X. Amatriain. Temporal diversity in recommender systems. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 210–217, 2010. 35

[21] L. Liu, N. Mehandjiev, and D.-L. Xu. Multi-criteria service recommendation based on user criteria preferences. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 77–84, 2011. 34

[22] N. N. Liu, X. Meng, C. Liu, and Q. Yang. Wisdom of the better few: cold start recommendation via representative based rating elicitation. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 37–44, 2011. 33

[23] H. Luo, C. Niu, R. Shen, and C. Ullrich. A collaborative filtering framework based on both local user similarity and global user similarity. *Mach. Learn.*, 72:231–245, 2008. 18

[24] T. Mahmood and F. Ricci. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, HT '09, pages 73–82, 2009. 9

[25] S. M. McNee, J. Riedl, and J. Konstan. Accurate is not always good: How accuracy metrics have hurt recommender systems. In *Extended Abstracts of the 2006 ACM Conference on Human Factors in Computing Systems (CHI 2006)*, 2006. 43

[26] P. Melville and V. Sindhwani. Recommender systems. In *Encyclopedia of Machine Learning*, pages 829–838. 2010. 28

[27] D. Mladenic. Text-Learning and Related Intelligent Agents: A Survey. In *IEEE Intelligent Agents*, pages 44–54, 1999. 22

[28] Y. Moshfeghi, B. Piwowarski, and J. M. Jose. Handling data sparsity in collaborative filtering using emotion and semantic based features. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 625–634, 2011. 34

[29] M. Nakatsuji, Y. Fujiwara, A. Tanaka, T. Uchiyama, K. Fujimura, and T. Ishida. Classical music for rock fans?: novel recommendations for expanding user interests. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 949–958, 2010. 36, 43

[30] P. Resnick and H. R. Varian. Recommender systems. *Commun. ACM*, pages 56–58, 1997. 9, 10

[31] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260, 2002. 33

[32] M. Sinnema and S. Deelstra. Industrial validation of covamof. *J. Syst. Softw.*, pages 584–600, 2008. 46

[33] K. Sugiyama and M.-Y. Kan. Serendipitous recommendation for scholarly papers considering relations among researchers. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, JCDL '11, pages 307–310, 2011. 42

[34] N. Tintarev and J. Masthoff. *Recommender Systems Handbook*. Springer US, 2011. 12, 17, 22, 24, 25, 26, 28, 29, 31, 32

[35] E. Toms. Serendipitous information retrieval. In *In Proceedings of the First DELOS Network of Excellence Workshop on Information Seeking, Searching and Querying in Digital Libraries*, pages 11–12, 2000. 38

[36] S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 109–116, 2011. 44

[37] Wikipedia. 15

[38] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 123–130, 2008. 35, 36, 37

[39] K. Zhou, S.-H. Yang, and H. Zha. Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 315–324, 2011. 34

[40] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 22–32, 2005. 34, 38, 41, 43, 46