User Behavior Learning in Designing Restaurant Recommender Systems: An Approach to Leveraging Historical Data and Implicit Feedback

by

Haoxian Feng

Thesis submitted in partial fulfillment of the requirements
For the Master of Applied Science in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

In typical restaurant recommendations, knowledge-based methods are used most often and do not take advantage of personal historical data. In this thesis, we are going to make some improvements to the Chicago Entrée restaurant recommender system. We will exploit the historical data and propose a weighted similarity approach to combine heuristic similarity with tag similarity between restaurants. Also, we show an improved way to mine the semantics of user behaviors using heuristic metric. These proposed approaches are evaluated by the comparison of three different pairwise approaches to learning to rank (LTR) in matrix factorization and five classic recommendation algorithms. The result shows that the combinatorial similarity outperforms the heuristic similarity on the precision, recall, F-score, and mean reciprocal rank.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Dr. Thomas Tran. His excellent guidance and continuous support have made my research fun and rewarding during my graduate study. He is the most erudite supervisor and gentle mentor who has encouraged me to overcome a lot of difficulties.

And last but not least I would like to thank my family and friends for their love and support that encouraged me from the very beginning to do my best to steadily step forward to gain my goal.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The emergence and prevalence of the Internet have led to the explosive growth of information which helps the world change from the era of information scarcity to the age of information overload [1][2]. In fact, both information consumers and information producers are facing significant challenges: for consumers, although information overload meets the user's demand for knowledge, it makes the consumers face the problem of how to choose the meaningful information when confronted with a large amount of data; for producers, it is hard to make the information or products famous and draw users attention. We need a tool to help us survive from these challenges.

1.1 Overview

1.1.1 What Is a Recommender System?

How do we buy things? How do we choose what articles to read or what music to listen to? How do we make decisions in daily lives? We find suggestions from our friends or relatives before making decisions. We read reviews about the products from other users online, and compare the specifications between similar products before we decide to buy them or not.

We are still using these old school skills to make decisions, but nowadays, people have started using recommender systems to solve these problems. Recommender systems' job is

to connect users and products. Not only do they help users find the valuable information or products but also they allow the information or products to be displayed to the targeted users. Usually, a recommender system connects user and items in three ways as represented in Fig.1.1.

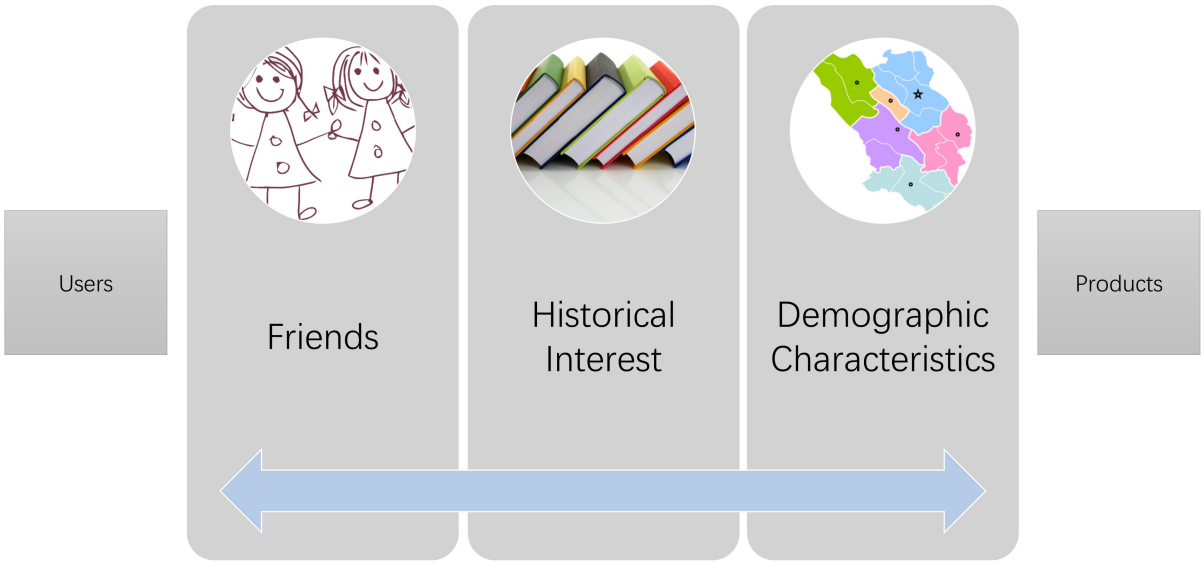

Figure 1.1: Three common ways to connect users and products.

In the Internet domain, a recommender system can provide online consumers with product information and recommendations through online E-business websites to enhance customers' shopping intention and improve the customers' shopping experience.

Its main roles are:

- Helping customers find attractive and high-quality goods in a short time, and enhance the users shopping experience.
- Increasing users' interest in the products.
- Reducing the time wasted in browsing repeated or fake information.
- Alleviating the side effects on the users that come from information overload.

- Collecting personalized information to recommend things that users really need.

Recommender systems' study has become a hot topic nowadays. Many researchers in academia, as well as industry, have been committed to the field of personalized recommendation. More and more contests in recommender systems with larger and larger rewards have appeared in both industrial and academic areas, such as Kaggle[3], Netflix Price[4], and Yahoo SIGKDD Cup [5]. More and more Internet companies, such as Amazon[6] and ebay[7], begin to compete in this field. We are all enjoying the benefits of recommender systems in helping us find better choices on doing almost everything.

1.1.2 Applications of Recommender Systems

In the 1990s, the field of electronic commerce began to flourish, and personalized recommender systems also rose their first wave. Some large e-commerce sites such as Amazon launched their personalized recommender system at that time. Literature [8] reported that 35 percent of Amazon's incremental sales came from the recommender system that was developed by themselves. At the same time, a sensational paper published by Amazon in 2000 entitled "Item-based collaborative filtering recommendation algorithms" [9] has now become one of the most famous literature in this area, and has a profound impact in both academia and industry. Ever since then, personalized recommendation technology has become an irresistible trend.

After 2000, the Internet was changed fundamentally which gave personalized recommender systems a new exploring stage. The first one is the rise of Web 2.0 where PC users become the key nodes in the Internet interactions. Individual users enjoy sharing their interests on the sites or in the apps which contain their individual needs. Compared to traditional Web 1.0, commercial sites pay more attention to the needs of users. Today, the vast majority of websites store users' information and create user profiles. Dedicated teams are organized to build large clusters containing dozens to hundreds of data nodes for analyzing users' needs [6, 7].

Another great change is the Internet has been integrated into our daily lives. Different

from the early Internet users, current users tend to use real-name authentication sites and applications to maintain the real social relationships and share real life tracks in their circle. Therefore, the information obtained by recommender systems carries more realistic scenario and emotional factors. The information not only provides recommender systems with substantial data support but also brings great business opportunities to the Internet companies.

Recommender systems is such a powerful tool that it can produce a significant positive effect on online sales, other than the achievement of sale boost in Amazon. Here are some facts [10]:

- Over 60% of videos watched by Netflix customers were recommended videos.
- 38% of click-through rates on Google News were recommended articles.
- 28% of people would like to buy more music if they find what they like on ChoiceStream.

1.2 Introduction of Entrée System

As we have mentioned above, in the era of the Internet economy, recommender systems have been widely used in various business areas, which also include food industry. The Chicago Restaurants Union has been using a recommender system called "Entrée", which is a find-me type system to provide restaurant recommendations to customers. This recommender system used a typical recommendation algorithm based on knowledge described as below [9].

1.2.1 Operation Process

The restaurant recommender system, Entrée, makes its recommendations by finding restaurants in Chicago that are similar to those users know and like. Users can navigate the system by stating their preferences about a given restaurant or about a restaurant selected

by the users within the database, thereby refining their search criterion in the system. This System, which has been used since 1996, behaved as a navigation mark of Chicago.

When using this system, firstly the user needs to submit an "Entry point" which is a known restaurant (or the system will give a default "Entry Point" for new users). Then, the system offers a range of criterion, and according to the criteria chosen by the user, it will display a restaurant that meets the user's need. At the same time, the user would rate the recommendation and set his/her preferences until an acceptable restaurant is recommended.

Here is an example of the Entrée type system in Europe, which performs the same as the original one in Chicago. As shown in Fig.1.2 [11], an Entrée-type system finds a restaurant called "Brauhof", which is similar to a restaurant called "Biegrasthof", which has been chosen by the user before.

Figure 1.2: Entrée-type system

Then, the user wants to find another restaurant that is similar to "Brauhof" but slightly cheaper. They can click the "Less$$" button, and the result will come out.

In the same way, the user can choose the cheaper, nicer or more traditional restaurant

by clicking the corresponding buttons. The system will refresh the recommendation results based on the users' operation, and finally help them find the most desired restaurants. The flowchart of this process is shown in Fig.1.3.

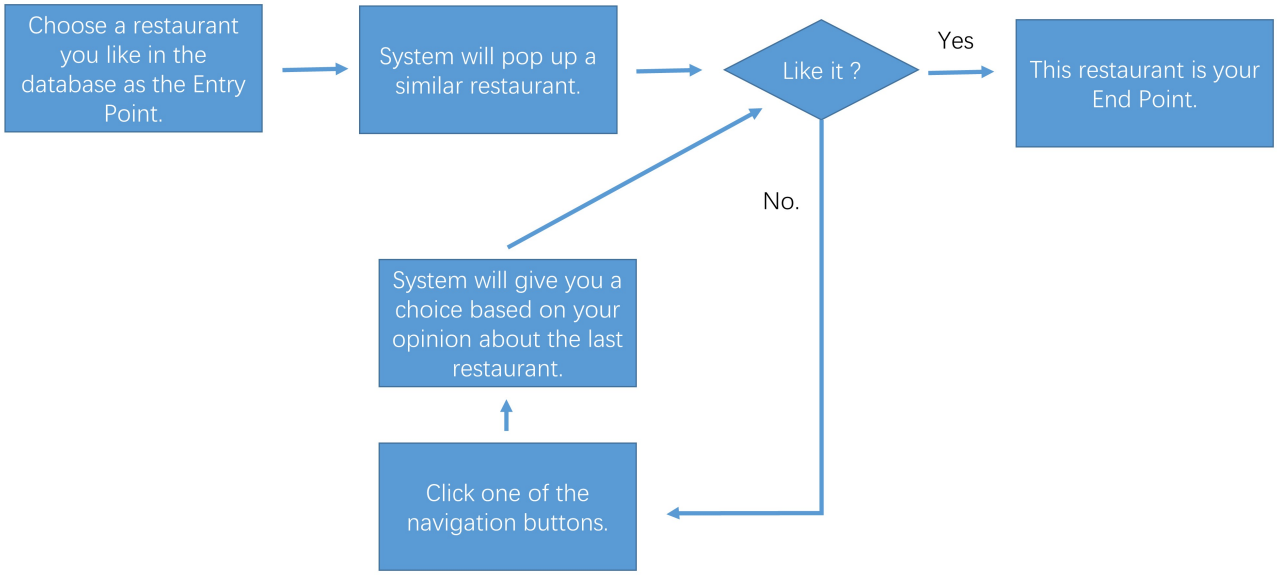

Figure 1.3: Entrée-type system's operation process.

In the design of the system, the most important consideration is how to provide a more humanized interaction between the website and users. Hence, the system will give users a few optimal options, rather than a long list of recommendations. Users can also balance the quality and price of the restaurant to find the best place to dine.

The recommendation technology that the Entrée system uses is knowledge-based searching. There are two basic search modes: one is based on the restaurants similarity, and the other relies on users' evaluation results. In the recommendation process, the user will select a given item category, and request for other similar items. For example, when a user is evaluating a restaurant A, if he/she clicks on the "Nicer" button, the system would recommend the restaurants that are superior to A to the user, discarding those relatively cheap and not very high-rated restaurants.

1.2.2 Motivations

The Entree system was a very successful and powerful tool and enjoyed great popularity among the tourists in Chicago. And this kind of traditional recommender systems use only a few online operations of users, such as cheaper and quieter, as a basis. In this case, this recommender system is far from meeting the real needs of users nowadays. The main reason for this problem is the insufficient exploitation of users' historical data.

Robin Burk [12] proposed the concept of hybrid recommendation model from the original knowledge-based recommendation and designed an improved version of the system named Entrée C. He firstly mapped the navigation information of the Entrée system to a series of implicit ratings, and user-based collaborative filtering algorithm (UBCF) was then used to get the prediction score. This method was proved to achieve good results.

Both Entrée and Entrée C systems use the dataset generated by the website, which generates one log for each interaction. Each line of the log contains the name of the restaurant that a user was browsing and the button that the user had clicked to acquire a new recommendation.

The response of Entrée system is completely determined by the example to which the user is responding and the specific critique given. For example, if a user responds to item A with the tweak "Nicer", the system determines the "niceness" value of X and rejects all candidates except those whose value are greater. Then, the website will pop up the restaurant with the greatest value. Therefore, Entrée system does not retain a user profile. If a recommender system does not retain users' profiles, it will be hard to make good recommendations. So the Entrée C system tried to solve this problem.

In Entrée C system, it uses all the data in one visit to make recommendations, which lasts from the "Entry point" to the "Exit point" or the webpage is closed by the user somehow. But Entrée C system still does not use all the historical data from each user, because this system treats every single visit as a new user. Moreover, the Entrée C system considers the relationship between behaviors based on some unreasonable assumptions, such as "More Traditional" and "Nicer", that are semi-opposite behavior pairs. For example, if a

user wants to dine in a vintage restaurant that has classical decoration and nice dishes, this user will respond to the "Entry point" with tweaks "Nicer" and "More Traditional" a lot of times. If we set these two behavior to be semi-opposite, it is really unlikely to provide a best-fitted restaurant for this user. These kind of behavior pairs will add noises into the similarity matrices. We will remove these unreasonable assumptions in section 3.3.2 and we will make some improvements on the similarity calculation in section 3.4.1.

1.2.3 Problem Statement

This thesis is going to address the problem that both Entrée and Entrée C systems are in lack of using historical data of each user and in lack of using relationships between user behaviors thoroughly. We are going to propose some techniques to analyze the relationship between user behaviors and historical data for dataset that can be used by the current recommender systems. In another word, these techniques can make more effective and accurate recommender systems.

We will introduce the mapping behaviors to scores process in section 3.1 and present a data scrubbing process in section 3.2.

It is important to value every single interaction with users because only one recommendation will be provided in one interaction. Users may get tired and lose patience and confidence in the system. Therefore, we want to provide the recommendation that the user likes with fewer interactions by improving the accuracy of recommending the restaurants that the user would rate positively. In other words, we need to improve the ability to tell the positives from the negatives. In this case, pairwise learning, one of the approaches of learn-to-rank methods, whose goal is to minimize average number of inversions in ranking, would be perfect for this goal.

1.3 Thesis Objectives

1. We will quantify both implicit and explicit feedback (as known as user behaviors) into ratings by analyzing the dataset of the Entrée system carefully. These ratings

can help us to understand the meaning of the behaviors much more intuitively.

2. We will try to figure out the relationship between different behavior and make use of the opposite behavior pairs in order to calculate users' similarities.

3. We will consider the similarity between restaurants and try to combine it with users' similarity to give better recommendations.

4. We will use several metrics to evaluate these approaches is appropriate for the Entrée dataset by comparing several different models.

1.4 Thesis Organization

This thesis consist of six chapters and is organized as below.

The first chapter is an introduction. In this chapter, we introduce both the current and historical status of personalized recommendation and the problems researchers have encountered in the development process. And then we focus on how the knowledge-based recommender system, Entrée, operates and how we can improve this system.

The second chapter is background and related work. In this chapter, we introduce both personalized recommendation technologies and the way to create a supervised learning model in recommender system. And we go through a pile of relevant papers to provide an overview of a lot of approaches in the area of restaurant recommendation.

The third chapter is methodology. We introduce the LightFM python library and the algorithms we used in the experiments. Also we propose the approach of how we leverage the historical data in the Entrée system.

The fourth chapter is evaluation. We use the four-fold cross validation in our experiment and use several metrics to evaluate the improvements of our proposed approach.

The fifth chapter is discussion which aims at addressing the value and potential applications of this thesis.

The sixth chapter is conclusion and future work which concludes the thesis and proposes some future research directions.

1.5 Summary

In this chapter, we introduce the motivation of the thesis and discuss the Entrée recommender system. We point out the weaknesses of this recommender system and state the motivation of this thesis. Then, we list our contributions in this thesis. Finally, we summarize our thesis chapter by chapter.

Chapter 2

Background Information and Related Work

In this chapter, we will introduce personalized recommendation technologies and present related theories of how to optimize a supervised learning model. Also, we will present a literature review in section 2.3.

2.1 Background Information

In this section, we will introduce personalized recommendation technology theory from collaborative filtering algorithm, which is one of the most popular recommendation methods, to context-aware models.

2.1.1 Recommendations Based on Collaborative Filtering

A common approach to recommender systems is collaborative filtering [15]. This approach has been studied extensively not only in academia, but also has been widely used in the industry. It can be divided into UBCF and item-based collaborative filtering algorithm (IBCF). We will talk about them in the next few sections.

2.1.1.1 User-based Collaborative Filtering (UBCF) Algorithm

UBCF is perhaps the oldest algorithm in recommender systems. It is no exaggeration to say that the birth of this algorithm marks the birth of the recommender system. The algorithm was proposed in 1992 and applied to the mail filtering system, which was used by GroupLens for news filtering in 1994 [16]. Since then until 2000, UBCF had been the most well-known and the most widely used algorithm in recommender systems.

In the real world, we always ask someone we trust or someone that shares similar interest for advice. For example, when we are freshmen in the university we may ask senior students in the same major about the choices of books and courses. The reason why we seek opinions from senior students is because we share the same interest in the course of study. This is the embryonic form of UBCF.

The implementations of UBCF can be simplified into three steps:

1. Find the group of people that share a similar interest or preference with the target user, which means that their rating matrices are similar.
2. Use the ratings of the group found in the previous step to predict how the target users would rate (or score) the same item.
3. Provide recommendations from high-rated (or high-scored) items.

In general, UBCF recommends the content that a user may be interested in according to the preferences of the group. Collaborative filtering analyzes the responses of users' preferences for goods, including responses such as "prefer" or "not prefer"[17].

Nowadays, the applications of UBCF are rarely seen. The most famous one is Digg [18]. Digg is a news website focusing on technology. In Digg's site, users can use the like and dislike buttons to rate articles. When user A clicks the like button, Digg thinks user A is interested in this article and willing to share this article with others. The website will find other users who had clicked the like button for the same article before, and then recommend the other articles that these users like to user A.

Digg has revealed the following effects since they used UBCF [19]:

- User feedback times increased by 40%.
- Each user would get 200 recommendations from 34 similar users averagely.
- Interactions between users increased by 24%.
- Users' review increased by 11%

Clearly, these are significant changes to Digg as a result of their using UBCF in making recommendations.

2.1.1.2 Item-based Collaborative Filtering (IBCF) Algorithm

IBCF is the most widely used recommendation algorithm in industries. In fact, Amazon, Netflix, Hulu, and YouTube are providing recommendations based on IBCF [11]. IBCF does not use the characteristics of items, but mainly focuses on using user's ratings to calculate the similarities between items.

The k-nearest neighbor algorithm [20] is usually used in IBCF. It computes the similarity between two items through the nearest adjacency matrix and then produces the predicted value for user u using the weighted average approach. The similarity calculation between items or users is an important part of collaborative filtering approaches. In addition, mechanisms such as Pearson correlation, cosine similarity, can be added into the algorithm to improve prediction accuracy. Then, the system will provide the most fascinating items for the targeted user according to the nearest neighbors' opinions [15, 16, 21].

The advantage of this method is that it is easy to interpret the recommendations by recommending items based on users' historical ratings, which helps users to understand why such item is recommended. If a user understands the reason of the recommendation and find it reasonable, he/she will trust the system (or website) and be a return user. IBCF does not require the consideration of the content of the items, and can easily be extended. But its shortcomings are also obvious: firstly, it needs users to rate items; secondly, when the data becomes sparse, its performance will become poor. Therefore, this method is not suitable for massive data mining and recommendation.

The main advantages of collaborative filtering (both user-based and item-based) are [22]:

- Being able to analyze some problems from the field of obscure concept using the experiences of others.
- Filtering out inquiries that are difficult for machine identification, such as art and music.
- Recommendations can be made to surprise users and inspire them to explore the potential preferences.
- Using the feedback from similar users to recommend a higher degree of personalized content.

The disadvantages of collaborative filtering are [22]:

- Cold start problem [23]. The system can not calculate the similarity between users or items if this is a new user or a new item with no rating record.
- Sparsity. In the real application, the number of users or items will be extremely large, which will make the rating matrix very sparse and even have zero similarity between two user.
- Scalability. Increasing the dimension of the rating matrix in the collaborative filtering approaches would result in a very complex computation and affect the systems scalability.
- System latency. The user's or item's similarity matrix need to be updated offline after adding new users or items, which would cause a latency for user to get recommendations.

2.1.2 Similarity Calculations

There are two popular methods to compute the similarities between users in collaborative recommender systems. One is a correlation-based approach [11]:

$$sim(x,y) = \frac{\sum_{f \in F_{xy}} (r_{x,f} - \bar{r_x})(r_{y,f} - \bar{r_y})}{\sum_{f \in F_{xy}} (r_{x,f} - \bar{r_x})^2 \sum_{f \in F_{xy}} (r_{y,f} - \bar{r_y})^2} \tag{2.1}$$

where the $r_{x,f}$ and $r_{y,f}$ are the ratings of the item f given by the user x and y, $\bar{r_x}$ and $\bar{r_y}$ is the average rating given by user x and y, and F_{xy} is the set of all items that is rated both by the user x and y.

Another one is a cosine-based approach [11]:

$$sim(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| * |\vec{y}|} \tag{2.2}$$

where the $\cdot$ means the dot product of vectors $\vec{x}$ and $\vec{y}$, $|\vec{x}|$ and $|\vec{y}|$ are the Euclidian length of the vectors.

Moreover, Jaccard similarity is also a popular method to calculate similarities [24].

2.1.3 Content-Based Recommendation Approach

Content-based recommendation (CBR) approach [25, 26] is the continuation and development of information filtering technology. This method takes full advantage of various features of users and items such as the author, the type and the publication time of a book, the age, gender, residence or other information of a user.

CBR generates users' and items' profiles respectively, and generates user preference model based on characteristics of users' rated items. And the system decides which item to be recommended based on item characteristics analysis. For example, in recommendations for books, a content-based recommendation system first analyzes the characteristics of books that has been purchased or evaluated by a user, and then recommends a book that is highly similar to those characteristics of the user's profile.

The content-based recommendation algorithm generally consists of three steps:

1. Tagging items: extract some of the characteristics of each item to represent the item.
2. Learning user preferences: utilize the characteristics of items that each user preferred or not to model user's preferences.

3. Generating recommendations: calculate the similarity between the user preference file and the characteristics of items, then similar items will be recommended to the user.

Content-based recommendation systems often use keywords to represent item characteristics and user preferences. The term frequency-inverse document frequency (TF-IDF) method is used to assign a weight to each keyword and represent the characteristics of the items and profile of users based on this keyword-weight vector [27]. The recommender can then use this vector to provide recommendations.

Here are the pros CBR [11]:

- No cold start problem.
- Performs well with sparse dataset.
- Recommendations are easy to be explained.

And the cons of CBR [11]:

- Require rich descriptions of items and well organized user profile.
- Can not surprise the user with serendipity recommendations.

2.1.4 Hybrid Recommender Systems

The hybrid model combines a variety of algorithms or models. This approach solves the deficiencies and limitations of a simple model and improves the prediction accuracy. The most important thing is that it solves the sparsity problem in some extend. However, it also increases the complexity of implementation. However, most of the commercial recommender systems have been using the hybrid model, such as Google news recommender system. Hybrid model can be divided into different types including feature weighted, mixed, switching, cascade and meta-level.

The pros and cons of hybrid models [11] are listed below:

Pros:

- Usually outperforms single models.
- No cold start problem.
- No popularity bias, can recommend items with rare features.
- Can implement serendipity and diversity.

Cons:

- It is difficult to get the right balance between different basic models.
- Lack of appropriate datasets.

2.1.5 Context-Aware Recommender Systems

This section focuses on context-aware recommender systems [28], which would consider context information such as time, location, and other factors when making recommendations. The traditional recommender systems only consider users and items (two-dimensional models).

$$R : User * Item \rightarrow Rating \tag{2.3}$$

But context-aware recommender systems that would take context information into account.

$$R : User * Item * Context \rightarrow Rating \tag{2.4}$$

Usually, we describe scenario information in a hierarchical structure, for example:

- Location: country, province, city, county.
- Time: Year, month, day.
- People: friends, heterosexual friends, girlfriend.

There are 3 main methods in a context-aware recommender system.

- Contextual pre-filtering, which selects only the information and data related to context for building recommender systems;
- Contextual post-filtering, which filters the results of recommender system and retains only the results that contain needed contextual information.
- Contextual modeling, which uses contextual information directly in the recommendation algorithm.

The flowcharts of the above methods are shown below in Fig.2.1 [29].

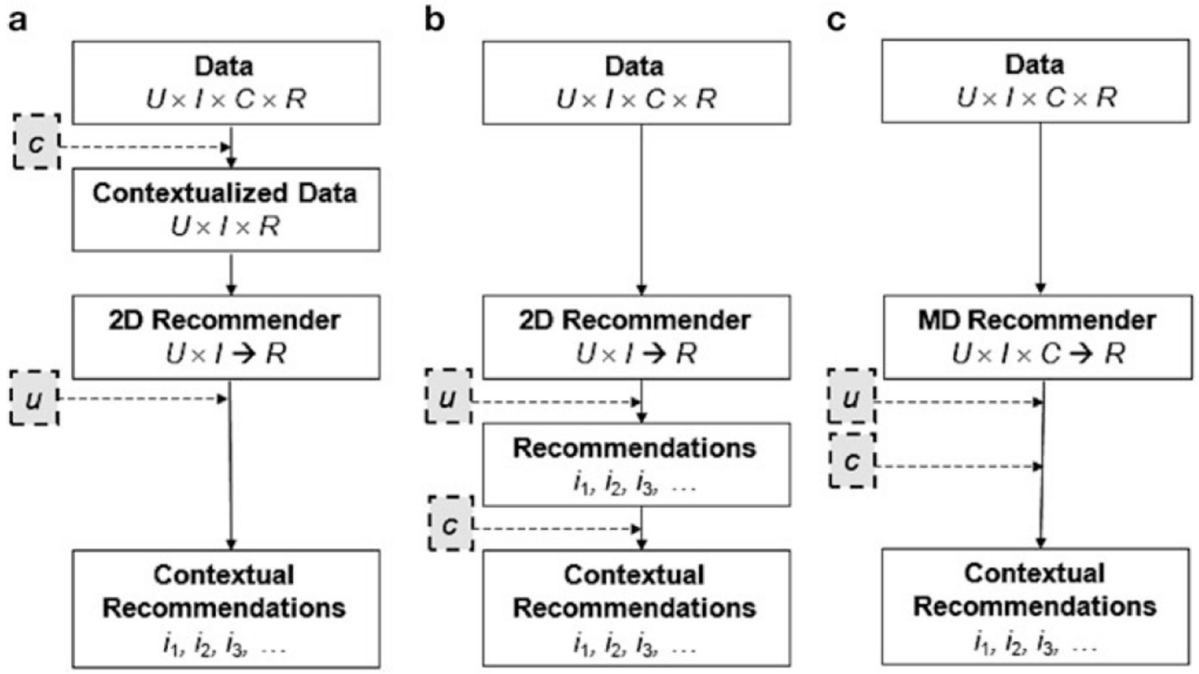

Figure 2.1: Paradigms for incorporating context in recommender systems. (a) Contextual pre-filtering; (b) contextual post-filtering; (c) contextual modeling

2.1.5.1 Contextual Pre-Filtering

Contextual pre-filtering uses context information to filter when choosing the data for recommender system. For example, if you recommend for a person who only surfs the Internet on Friday, you simply filter and get the data of all users on every Friday to construct the recommendation model as the following equation:

$$Rating \quad matrix \quad = R_{user*item}^{D[Time=t](user,item,Rating)}(u,i), \forall (u,i,t) \in U * I * T \tag{2.5}$$

where u is the user, U is the set of users, i is the item, I is the set of items, t is one of the days in the week, T is the set of days in the week, $U * I * T$ is a three dimensional matrix, $D[Time = t]$ is the day of the week, and R is a $user * item$ rating matrix that contains the ratings of the certain day in the week.

The above-described method in equation (2.5) and (2.6) is known as exact pre-filter, which uses the current context-aware information to make a match. However, it has some disadvantages. For example, too accurate contextual information may not be practical enough. It makes no difference whether a user goes to the cinemas on Saturday or Sunday, but it does make a difference on Wednesday (weekdays). Therefore, when filtering the information, Sundays data should not be filtered out.

Moreover, the amount of data after exact filtering is relatively small, hence the problem of sparsity will become more notable. In general, it can be solved using contextual generalization processing which uses a more generalized information ($Time \in S_t$, where S_t is known as contextual segment) instead of using the original context information ($Time = t$). Equation (2.5) will be transformed into equation (2.6).

$$R_{user*item*Times}^{D}(u,i,t) = R_{user*item}^{D[Time \in S_t](user,item,A(Rating)}(u,i), \tag{2.6}$$

where $A(Rating)$ means the average of all the ratings that user u has rated in time period S_t.

2.1.5.2 Contextual Post-Filtering

Contextual post-filtering will not consider the contextual information when inputting the data and modeling until the recommended item list is generated. It will consider the context information and do the following process:

- Filtering out irrelevant items.
- Adjusting the order of items in the list.

This process is depicted in Fig.2.2 [29]

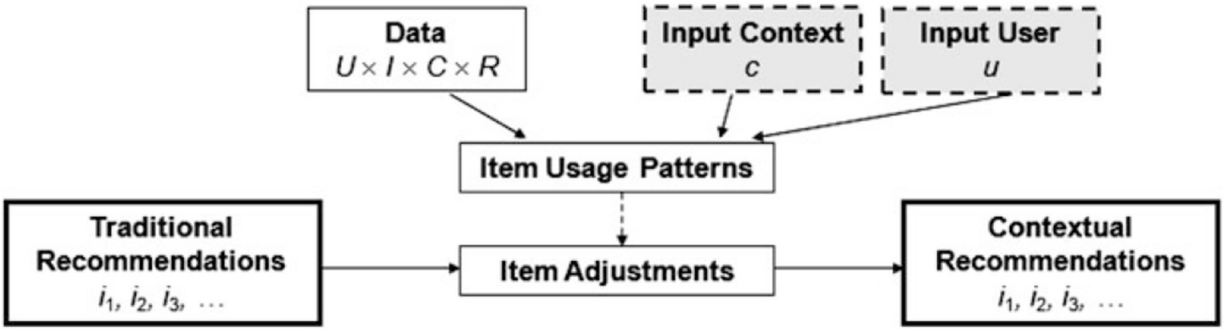

Figure 2.2: Contextual post-filtering approach: recommendation list adjustment.

Contextual post-filtering can be divided into two types: heuristic-based and model-based approaches.

- Heuristic-based approach filters out items containing small amount of common features and changes the order of recommendation according to the number of common features between the item and the current context information.
- Model-based approach builds a model to estimate the probability that a user selects a certain item in a certain context by filtering out the items with low probability and sorting the recommendation list according to multiplicative results of the predicted score and probability.

2.1.5.3 Contextual Model

The contextual model uses context information to construct recommendation function and then directly takes the recommendation results of the function as a certain score to provide recommendations . Although contextual pre-processing and contextual post-processing can be realized by the two-dimensional recommendation function, the contextual model can be extended to higher dimensional recommendation model. This kind of multidimensional model can successfully fuse contextual information into the predictive model or heuristic computation including decision trees, regression model, and probability model. The formula is expressed as:

$$Rating = R(User, Item, Context) \tag{2.7}$$

In the past 10–15 years, a considerable number of recommendation algorithms based on heuristic or predictive modeling techniques have been extended from two-dimensional model to the multidimensional model.

2.2 Basic Theory: Creating a Learning Model

In most cases, a recommender system would have several necessary parts. For example, a context-aware recommender system would include the context-aware model and user profile. They will coordinate with the core algorithm and learning rules to provide recommendations.

Fig.2.3 below depicts the architecture of a personalized recommendation system, which can be separate into four stages. Firstly, we can apply context factors into the context-aware approach and use users' information (demographical and geographical) to build user profile. Secondly, we can make some learning rules based on corresponding service strategy. Thirdly, we put our data into the core algorithm and provide recommendations. Finally, we can modify those rules and algorithm by considering the feedback from users or the offline experimental results.

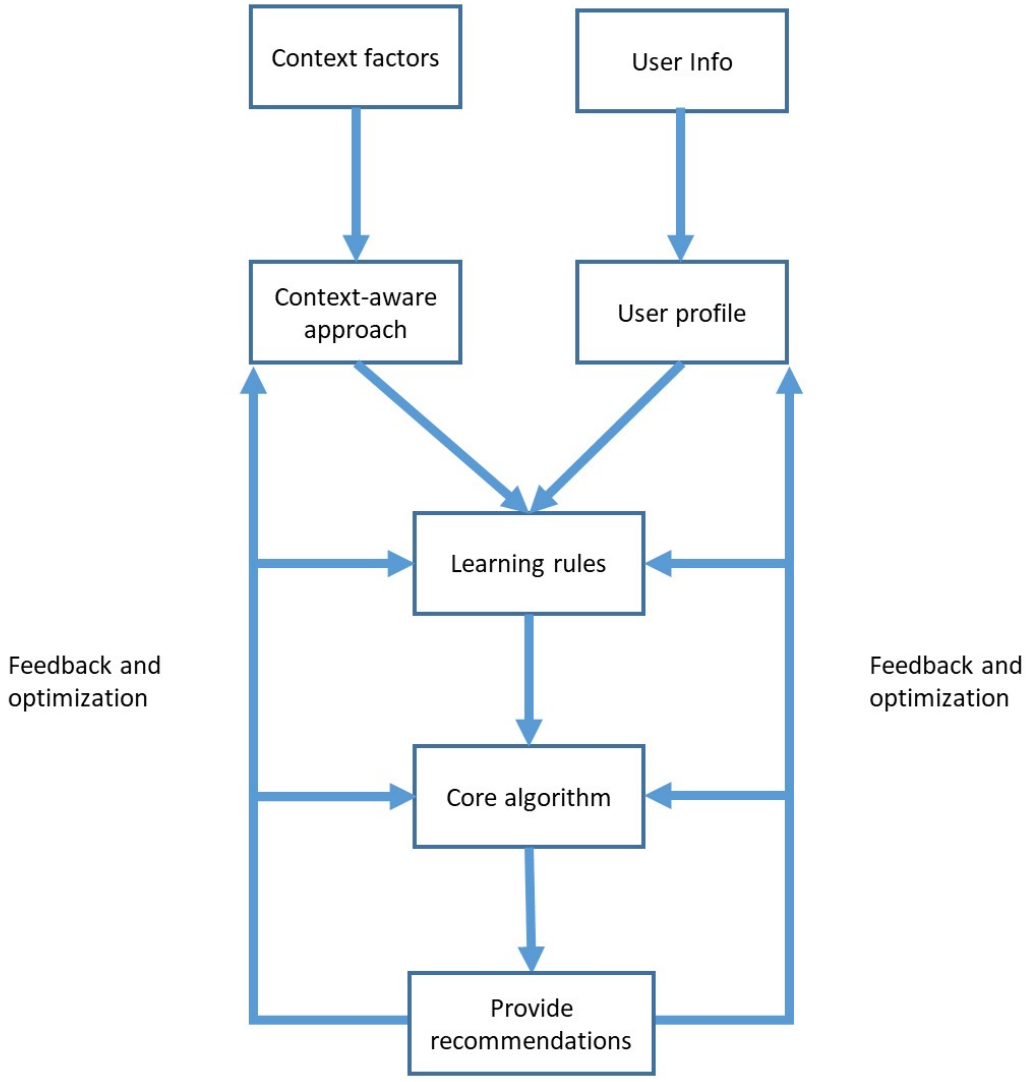

Figure 2.3: Architecture of personalized recommendation system.

The core algorithm usually contains an objective function we need to minimize and the

process of doing that. Loss function, regularization terms, and gradient descent process are the common combination to form the core algorithm. We will introduce them in the next few subsections.

2.2.1 Loss Function

In statistics, a loss function is a measurement of the losses and errors (that are related to "false" estimates, such as the cost or loss of an event) while in machine learning it is used to evaluate the margin between the values assigned by the scoring classifiers or systems (the predicted value) and the samples' real values. We map each sample's margin to an associated loss function [30]. A loss function is used in almost every single machine learning model.

The most widely used loss function is sigmoid:

$$L(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

It has a characteristic "S"-shaped curve, and has been widely used in machine learning as well as in artificial neural networks.

Also, we have the hinge loss as follows:

$$L(x) = max(0, 1 - tx), \quad t = 1 \quad or - 1 \tag{2.9}$$

This function has been widely in support vector machine (SVM), which is good at dealing with high-dimensional data.

And the heaviside function, known as 0-1 loss, is a step function.

$$L(x) = \begin{cases} 0, & x < 0 \\ \frac{1}{2}, & x = 0 \\ 1, & x > 0 \end{cases} \tag{2.10}$$

Fig.2.4 shows the curves of the three different loss functions mentioned above.

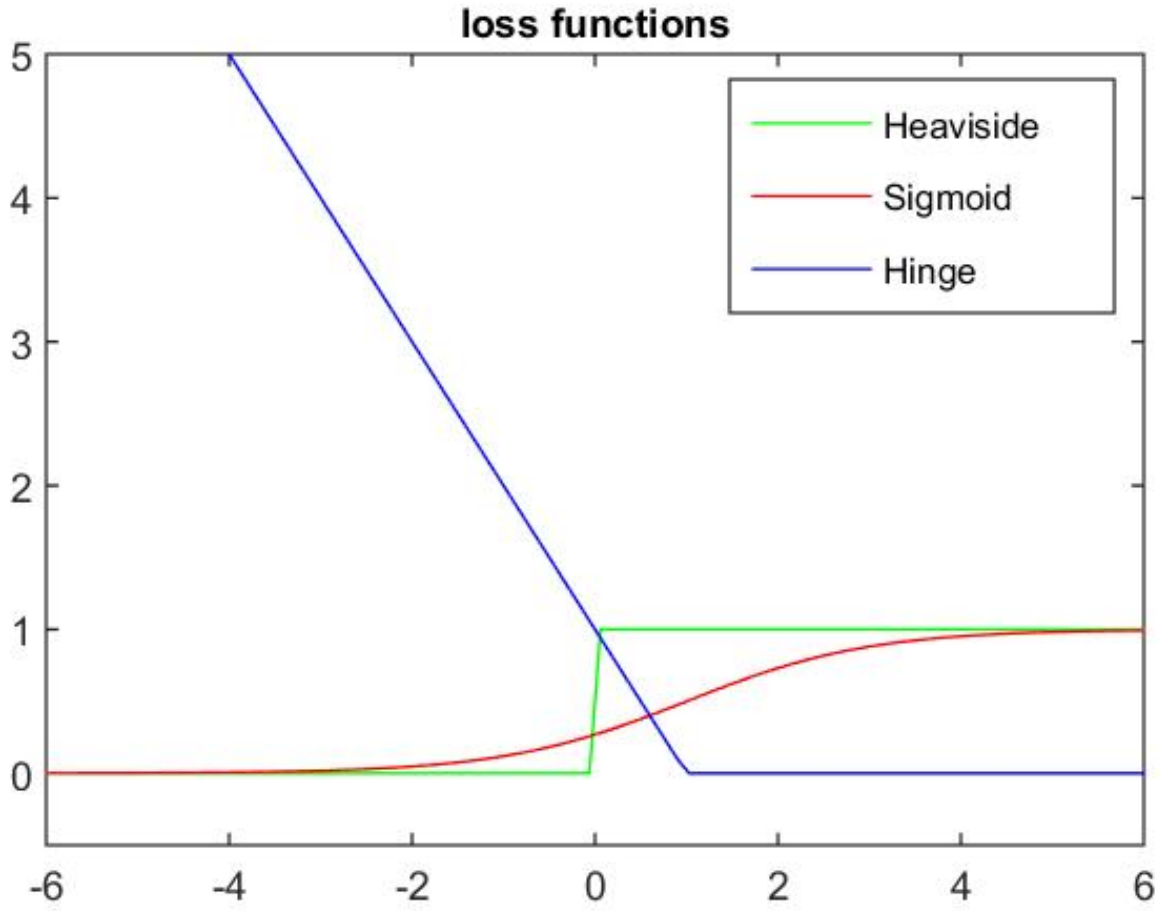

Figure 2.4: Three common loss function.

There are more types of loss functions. Although loss functions can be customized in practice, researchers need to consider what optimization method to be used to minimize the loss. Also, regularization terms will be involved in the real optimization function, for the feature selection and to prevent the overfitting problem.

2.2.2 Regularization

Regularization is a general method to avoid the overfitting problem by applying additional constraints to the weight vector. A common approach is to add a regularization term to the weight. In supervised machine learning area, problems can always be transformed into minimizing your error while regularizing your parameters. Minimizing the error is to make

our model fit our training data, and regularizing parameters is to prevent our model suffer from overfitting the training data. Too many parameters will lead to a more complex model, which would easily overfit the training set and yield a small training error.

Minimizing the training error is not our final goal, but minimizing the test error is, which means the model can provide accurate predictions with new data. We need to ensure that we minimize the training error based on a simple enough model so that the parameters we get have a good generalization performance (that is, the test error is small).

Usually, an objective function in supervised machine learning can be defined as:

$$J(\theta)^* = \underset{\theta}{argmin} \sum_i L(y_i, f(x_i; \theta)) + \lambda\Omega(\theta) \tag{2.11}$$

where $J(\theta)^*$ is the objective function and θ is the parameter set of this objective function, $L(y_i, f(x_i; \theta))$ is the loss function that evaluates the margin between predicted and the real value of the item i. We want this term to be as small as possible because we are trying to fit the training data. To minimize the test error, we need the second term $\Omega(\theta)$ which is the regularization term of parameter θ while λ is the parameter of the regularization term.

There are many options for the regularization term $\Omega(\theta)$, in most cases it is a monotonically increasing function of model's complexity. The more complex is the model, the greater is the regularization term's value. For example, the regularization term can be the norm of the model parameter vector. However, different choices have different effects on the parameter θ, and produce different results. The most common norms are: zero norm, L1 norm, which is the sum of the absolute differences between the target value and the estimate value, L2 norm, which is the sum of the square of the differences between the target value and the estimate value, trace norm (or nuclear norm), and Frobenius norm, etc.

We can create an objective function properly with an appropriate loss function and one or a few regularization terms. In order to find the hyper-parameters when this loss function reaches its minimum, we need use the gradient descent method.

2.2.3 Gradient Descent

In order to minimize the loss function, we need to use an optimization method. The most widely used method is gradient descent.

It was originally used in linear regression and then expanded to different machine learning algorithms. This method can help us to find the minimum values of functions. Gradient descent has three different version:

- Batch gradient descent (BGD)
- Stochastic gradient descent (SGD)
- Mini-batch gradient descent (MBGD)

Its goal is to minimize the objective function $J(\theta)^*$, where θ is the parameter set of the learning model. Gradient descent will update the parameters in the opposite way of the $\theta's$ gradient in every iteration for every parameter θ. A learning rate determines the minimum iteration times for the function to reach its local minimum.

SGD is the most widely used method in the field of machine learning, and it usually consists of the following steps.

Suppose we have the objective function as follow:

$$J(\theta)^* = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 \tag{2.12}$$

We compute its partial derivative of θ:

$$\frac{\partial J(\theta)^*}{\partial \theta} = -\frac{1}{2}(y^i - h_\theta(x^i))x_j^i \tag{2.13}$$

Since we need to minimize this function, we update the hyper-parameter θ based on the direction of $\theta's$ gradient descent by using one sample per update:

$$\theta_j^* = \theta_j + (y^i - h_\theta(x^i))x_j^i \tag{2.14}$$

In SGD, θ is updated once for each sample, so we can achieve the optimal solution with only a few thousands or tens of thousands of samples even if we have millions of them.

2.2.4 Learning Rate Schedule

There are several different optimization methods for the gradient descent such as: Momentum, Nesterov, Adagrad, Adadelta, RMSprop and Adam.

They are the solutions to either one or both disadvantages of simple gradient descent. The first disadvantage is that it is hard to choose a proper learning rate. The second one is that it is not good to use only one learning rate to update all parameters.

This example utilizes two different learning rate schedules: Adagrad [31] and Adadelta [32]. Both of them are trying to optimize the original gradient descent method.

2.2.4.1 Adagrad

Adagrad is an optimization learning method for gradient descent. It can self-adapt different learning rates to each parameter, and this ability is its main advantage. Also, Adagrad applies larger updates on sparse features and smaller updates on non-sparse features. With this special process, it is suitable for dealing with a sparse dataset.

Dean et al.[33] have found that Adagrad improved the robustness of SGD greatly. And Google has used it to train large-scale neural networks which have a lot of functions including recognizing cats in Youtube videos. Moreover, Pennington et al.[34] used Adagrad to train GloVe word embeddings, which is an algorithm of word vector representations, because frequent words need much smaller updates than infrequent ones.

Now, we will introduce how Adagrad works. We set $g_{t,i}$ to be the gradient of the objective function at the time step t.

$$g_{t,i} = \nabla_\theta J(\theta_i) \tag{2.15}$$

where the ∇_θ is the operator to calculate the gradient of $J(\theta_i)$.

The regular SGD update rule is:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i} \tag{2.16}$$

Adagrad will do some amendments to the learning rate η at each time step t for every parameter based on the past gradient for θ_i.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i} \tag{2.17}$$

where $G_t \in \Re^{d \times d}$ is a diagonal matrix, the diagonal element i is the sum of the squares of the gradient of parameter θ_i, and ϵ is a smoothing term to avoid the denominator being zero.

We vectorize our equation by performing an element-wise matrix-vector multiplication $\odot$ between G_t and g_t.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \odot g_{t,i} \tag{2.18}$$

The main disadvantages of Adagrad are we need to sum up all the squared gradients in the past sequence, and the changes in the learning rate will drop to a very small value gradually which makes it consume more time to find the local minima. The Adadelta learning rate schedule aims to solve this problem.

2.2.4.2 Adadelta

Adadelta is an extension of Adagrad that restricts the number of past gradients to be accumulated rather than summing up all past squared gradients. That makes the loss function converge quickly.

Adadelta does not store the previous squared gradients. The sum of them is defined as the decay average of all past gradients. If we assume the running average at time t is $E[g^2]_t$, we can compute:

$$E\left[g^2\right]_t = \gamma E\left[g^2\right]_{t-1} + (1-\gamma) g_t^2 \tag{2.19}$$

where γ is a decay constant. For clarity, we rewrite the SGD update rule to the parameter update vector $\triangle\theta_t$:

$$\triangle\theta_t = -\eta \cdot g_{t,i} \tag{2.20}$$

$$\theta_{t+1} = \theta_t + \triangle\theta_t \tag{2.21}$$

Therefore, we can change the parameter update vector of Adagrad in equation (2.18) to:

$$\triangle\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{2.22}$$

And then we replace the G_t with $E[g^2]_t$:

$$\triangle\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{2.23}$$

Since the denominator is just a type of root-mean-squared (RMS), we rewrite it as:

$$\triangle\theta_t = -\frac{\eta}{RMS\left[g\right]_t} g_t \tag{2.24}$$

where a constant ϵ is added to better condition the denominator. The parameter update function is:

$$\triangle\theta_t = -\frac{\eta}{RMS\left[g\right]_t} g_t \tag{2.25}$$

In order to solve the problem of different hypothetical units in the updates, we use squared parameter updates instead of squared gradients':

$$E\left[\triangle\theta^2\right]_t = \gamma E\left[\triangle\theta^2\right]_{t-1} + (1-\gamma)\triangle\theta_t^2 \tag{2.26}$$

So we have the RMS function as:

$$RMS\left[\triangle\theta\right]_t = \sqrt{E\left[\triangle\theta^2\right]_t + \epsilon} \tag{2.27}$$

Since the $\triangle\theta_t$ is unknown for the current time step, we replace the learning rate η in the previous update rule with $RMS\left[\triangle\theta\right]_{t-1}$ to approximate it. Finally, we have the Adadelta update rule:

$$\triangle\theta_t = -\frac{RMS\left[\triangle\theta\right]_{t-1}}{RMS\left[g\right]_t} g_t \tag{2.28}$$

$$\theta_{t+1} = \theta_t + \triangle\theta_t \tag{2.29}$$

From equation (2.28), we can see that we do not need to specify an initial learning rate because Adadelta can process the update without the default learning rate.

For different datasets, there is no clear winner between these two schedules (Adagrad and Adadelta). We will show how we choose the one that achieves better results in Chapter 4. In the next few subsections, we will discuss the learn to rank methods and matrix factorization approach that are used in ranking the recommended restaurants.

2.2.5 Matrix Factorization Methods

The user-item rating matrix can be massive in the real world application with hundreds of thousands of users and items. Since a user can never rate all the items, the user-item matrix will contain a lot of null value which makes it a sparse matrix. The core of matrix factorization believes user interest is only affected by a few factors and we can factorize this sparse high-rank user-item rating matrix into two low-rank matrices to learn those few factors.

Although this method came out as singular value decomposition (SVD) early in 1998 [56], no one thought it could be a powerful tool for recommender systems until 2006. In 2006, Simon Funk proposed Funk-SVD algorithm on his blog which was used by the Netflix Prize winner Koren later on [57]. After that, matrix factorization algorithm became a hot topic in the field recommender systems.

The mathematical foundation of the matrix factorization algorithm [58] is the transformation of matrices. According to linear algebra, row operations of matrix A can be represented by a matrix product pA, while column operations can be represented by a matrix product Aq. Therefore we have $A = pEq = pq$, where E is a diagonal matrix.

In recommendation study, the basic idea is decomposing the rating matrix $R \in \mathbb{R}^{m \times n}$ into the matrix product of user factors' matrix $U \in \mathbb{R}^{m \times k}$ and item factors' matrix $V \in \mathbb{R}^{n \times k}$ which can be represented as:

$$R \approx U \times V^T \tag{2.30}$$

A more vivid representation of this process is shown in Fig.2.5.

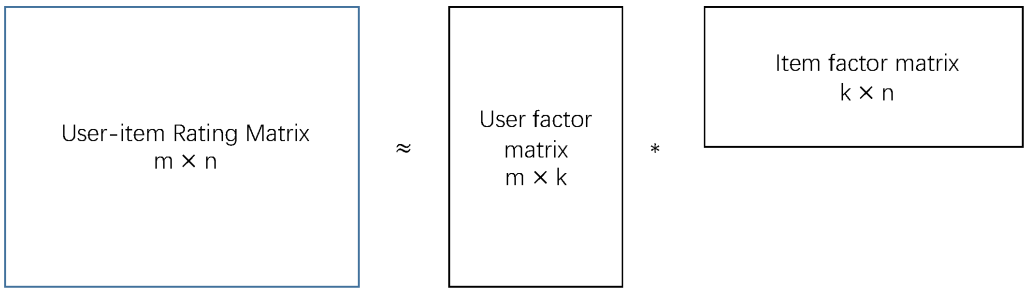

Figure 2.5: Matrix factorization model[59].

Simply put, it is decomposing the high-rank user-item rating matrix into two low-rank factor matrices. Because this two factor matrices store the feature factors of all users and items, matrix factorization deducts the dimension differently from principal component analysis (PCA) [60].

Our goal is to find those two matrices on the right side by using existing ratings. The rating given from user i to item j is $r_{ij} = u_i \dot{v}_j$, and it can also be represented by $r_{ij} = f(U_i, V_j)$ in a more general situation. More intuitively, in this section, we consider the rating matrix as the dot product of the user factor matrix and item factor matrix. Let us assume that the difference between the real rate and the predicted rate from user i to item j is Gaussian distribution. Based on this assumption, we can derive our loss function from:

$$R_{ij} \leftarrow U_i^T V_j \tag{2.31}$$

And according to the Gaussian distribution, we have:

$$
\begin{aligned}
p(R_{ij}|U_i^T V_j), \delta^2) &\rightarrow p(R|U, V, \delta^2) \\
&= \prod_{i=1}^{N} \prod_{j=1}^{M} [N(R_{ij}|U_i^T V_j, \delta^2)] \\
&\rightarrow argminlnp(R|U, V, \delta^2) \\
&= -\frac{1}{2\delta^2} \sum_{i=1}^{j=1} (R_{ij} - U_i^T V_j)^2 - \frac{1}{2}[\sum_{i=1}^{N} \sum_{j=1}^{M} (ln\delta^2 + ln2\pi)] \\
&\rightarrow argmax \sum_{i=1}^{N} \sum_{j=1}^{M} (R_{ij} - U_i^T V_j)^2
\end{aligned}
\tag{2.32}
$$

In order to find the maximum value of $R_{ij} - U_i^T V_j$, we transform this to be the loss function as follows:

$$
L = R - U \times V^T \tag{2.33}
$$

We can use gradient descent to find the U and V that make the loss L converge at the minimum value. Thus we have:

$$
R \approx U \times V^T = \hat{R} \tag{2.34}
$$

And we can use $\hat{R}$ to make recommendations for users.

The positive sides of using matrix factorization are:

1. Matrix factorization can be programmed quickly, and the model can be trained by each iteration of stochastic gradient descent.

2. Matrix factorization can be run with relatively low-level space and time complexity. Projecting the high-rank matrix into two low-rank matrices saves a lot of storage. Although the model building time would be long and can only be done offline, this algorithm is still suitable for real-time recommendations. Because after reducing the dimensionality of user-item rating matrix, the computation complexity is decreased significantly, which means the computational time is reduced rapidly.

3. Matrix factorization can produce an accurate prediction. Usually, the accuracy is higher than collaborative filtering algorithms and content-based recommender systems [61].

4. Matrix factorization has a good expansibility. We can add other factors such as implicit feedbacks [62]or timeline [63] to become SVD++ algorithm.

The negative sides of using matrix factorization are:

1. The fitting result is not explainable because when we decompose matrix R to matrices into U and V, their features can not be clarified with a general concept and we can only treat them as latent semantic space.

2. In other words, matrix factorization process is sort of a process of clustering. Every feature we have equals to a cluster but we do not have a general way to name these clusters.

In real world applications, we can use several distinguishing characteristics to name these clusters. For example, in news recommendation, we can use keywords to name different types of news.

2.2.6 Learning to Rank Algorithms

The study of recommendation systems can be divided into two main directions:

- Ratings prediction.
- Top-k recommendations.

Original memory-based method (such as UBCF and IBCF) and model-based method (such as pure matrix factorization, biased matrix factorization and probability matrix factorization) are the solutions to the rating prediction by optimizing the error between predict and real ratings, such as the root-mean-square error (RMSE). But in most cases,

users want to have a list of recommendations in an organized order rather than having some ratings only. Especially, users focus mostly on the first few items on the recommendation list. If we want to use these predictions to produce top-k recommendation list, we need to sort these ratings in descending order.

But high rating prediction accuracy (or a small enough error in RMSE) does not guarantee a good ranking list. In this case, we should optimize the top-k list directly by using the learn to rank algorithms to give users a better experience when using the recommender system.

Learn to rank methods are widely used in information retrieval to find the best ranking orders among documents. These approaches use the pair features $< query, document >$ (Q-D pair) as input and predict the relevance of new Q-D pairs to produce a document list in proper order for a particular query. Analogously, in personalized recommendation area, the goal is to give the user a list of interesting items, and we can treat user-item pairs as the Q-D pairs in order to use the learn to rank methods. The flowchart of learn to rank methods is shown below in Fig.2.6

In the information retrieval study, the learn to rank method needs explicit document features while in personalized recommendation study, neither users and items have explicit inputs. We usually only have user-item rating records as the input data, so the learn to rank methods using in recommender systems are different from the methods used in information retrieval. There are three approaches of learn to rank methods:

- Point-wise approach
- List-wise approach
- Pair-wise approach

We will introduce the first two approaches briefly and discuss the third approach deeply as it is the core approach that we use in this thesis.

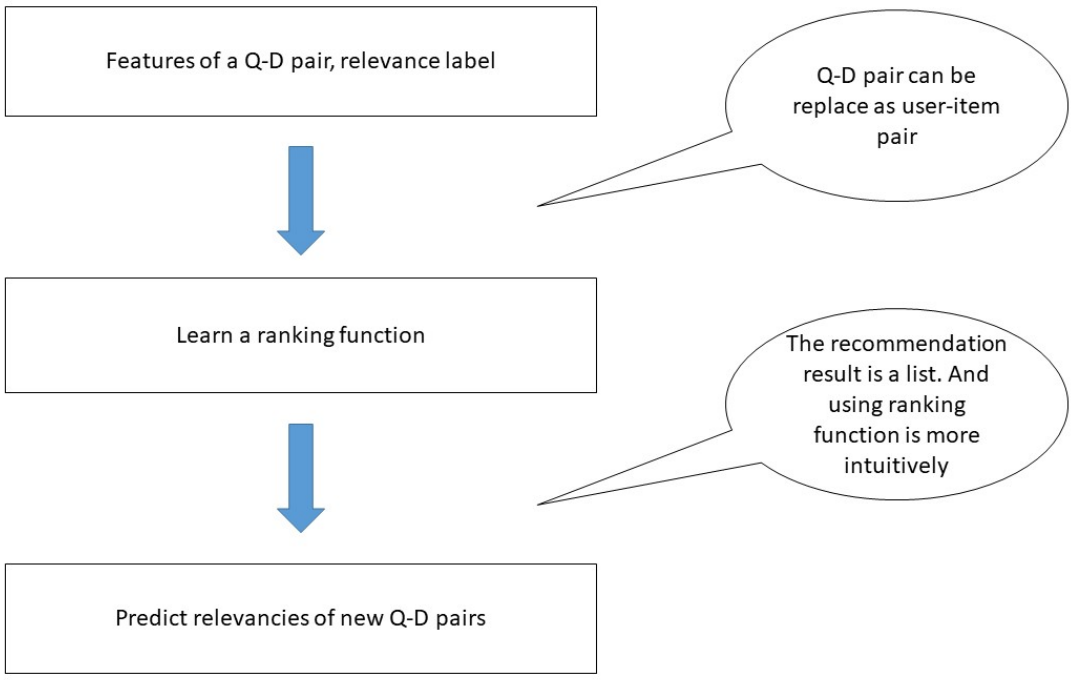

Figure 2.6: Flowchart of learn to rank algorithms.

2.2.6.1 Pointwise Approach to Learning to Rank

Pointwise approaches to learning to rank only consider the accuracy of single item's predicted rating and return a list of recommendations by sorting items' predicted ratings in descending order. For example, the collaborative filtering approach, the SVD++, and logistic regression can be categorized as pointwise approaches. This kind of algorithm does not consider the relative order between items, therefore it does not optimize the items' ranks directly.

2.2.6.2 List-wise Approach to Learning to Rank

We can also try to directly optimize the ranking of the whole list by using a listwise approach. RankCosine [64], for example, uses similarity between the ranking list and the ground truth as the loss function. ListNet [65] uses KL-divergence as loss function by defining a probability distribution. RankALS [66] is a recent approach that defines an

objective function that includes the ranking optimization, which is called alternating least squares (ALS).

2.2.6.3 Pairwise Approach to Learning to Rank

There is a growing research effort in finding better approaches to ranking. The pairwise approach to ranking, for instance, optimizes a loss function defined as pairwise preferences from the user. The goal is to minimize the number of inversions in the resulting ranking. It can be represented as Fig.2.7:

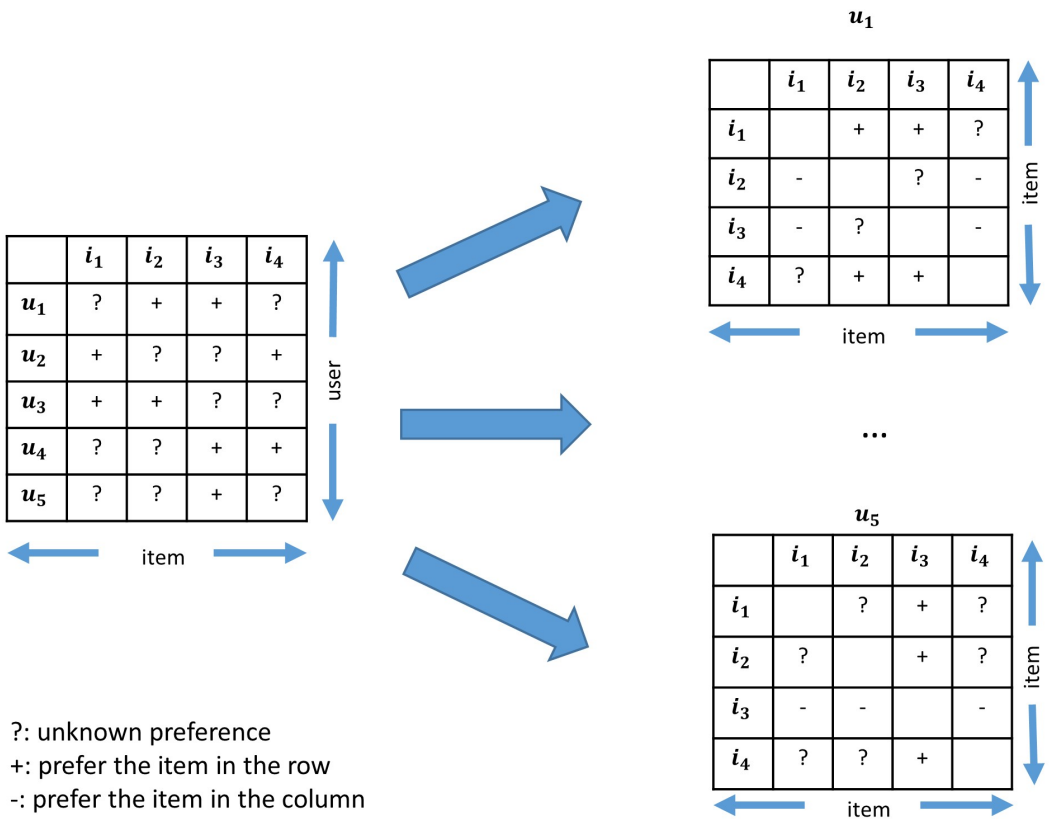

Figure 2.7: Example of pairwise approach to learning to rank (row to column).("?" means unknown preference, "+" means users prefer the item in the row and "-" means users prefer the item in the column)

The left-hand side of the Fig.2.7 represents five different users' preferences on four different items, symbol "+" means user like this item, and symbol "?" indicates unknown preference. The figure represents different users' preferences separately on the right-hand side, where "+" means user like this item in this row more than the item in this column, "−" means dislike or less preference and "?" means unknown preference.

This kind of algorithm is good at adjusting the relative order between different items, which means they are also good at dealing with dataset full of implicit feedback. Pairwise learning algorithms, which is known as the preference learning, consider two samples in each iteration. These algorithms focus on distinguishing the relative order between two samples and uses the relative order between items to compute a whole item recommendation list in descending order. We will talk about three different examples in the following subsections.

2.2.6.3.1 Bayesian Personalized Ranking pairwise loss (BPR) So far, there are several methods applied to the recommender systems based on implicit feedback. Although both pure matrix factorization and pure collaborative filtering can deal with implicit feedback, neither of them are implemented with ranking optimization. Steffen Rendle, Christoph Freudenthaler et al. proposed a general optimization method called BPR_Opt [67, 68], and it was used in personalized ranking . This method gets the maximum value of the posterior estimation by Bayesian analysis and uses stochastic gradient descent based on bootstrap sampling [69] to provide a general learning model.

BPR_Opt's optimization goal is to maximize joint probability by ranking the items with feedback higher than those without [70]. Also, BPR can be used in matrix factorization and collaborative filtering algorithm to help them achieve personalized ranking. Steffen Rendle used bootstrap sampling [69] to update the model incrementally because there were a lot of pair-samples, and AUC [71] was used to evaluate the performance of the model, which calculates the ratio of correct orders. Moreover, the results indicate this model performs better than HuYifan's collaborative filtering model [72]. Fig.2.7 shows that the core of BPR is to put a higher preference on the observed values than those that are unobserved.

Let us suppose X is the user-item preference matrix, x_{ui} is user $u's$ preference level to

item i. The matrix factorization model for X is:

$$\hat{X} = WH^T \tag{2.35}$$

BPR uses posterior estimation to find out the parameter W and H. According to Bayesian function, we have:

$$p(\theta|>_u) \propto p(>_u|\theta)p(\theta) \tag{2.36}$$

where $>_u$ represents the total order from user u to different items, θ represents either the parameter W or H.

And we can assume $W's$ and $H's$ posterior estimation satisfied normal distribution with zero average value, then we have:

$$BPR\_Opt = \ln p(\theta|>_u) = \sum_{(u,i,j)\in D_s} \ln \sigma(\hat{x}_{uij}(\theta)) + \lambda \left\|\theta\right\|^2 \tag{2.37}$$

where $\hat{x}_{uij}(\theta)$ is an arbitrary real-valued function of the model parameter vector which captures the special relationship between user u, item i and item j. For convenience, in the following we will skip the argument θ from $\hat{x}_{uij}(\theta)$. And $\sigma(x) = \frac{1}{1+e^{\ x}}$.

We use the most common loss function Sigmoid as the example of derivation as shown in equation (2.38).

$$\begin{aligned} \frac{\partial BPR\_Opt}{\partial \theta} &= \sum_{(u,i,j)\in D_s} \frac{\partial}{\partial \theta} \ln \sigma(\hat{x}_{uij}) + \lambda \frac{\partial}{\partial \theta} \left\|\theta\right\|^2 \\ &\propto \sum_{(u,i,j)\in D_s} \frac{-e^{\hat{x}_{uij}}}{1+e^{\hat{x}_{uij}}} \times \frac{\partial}{\partial \theta} \hat{x}_{uij} + \lambda\theta \end{aligned} \tag{2.38}$$

According to the definition of $\hat{x}_{uij}$, we can get its derivatives (2.39) as follow:

$$\frac{\partial}{\partial \theta} \hat{x}_{uij} = \begin{cases} h_{if} - h_{jf}, & if \quad \theta = w_{uf} \\ w_{uf}, & if \quad \theta = h_{if} \\ -w_{uf}, & if \quad \theta = h_{jf} \\ 0, & otherwise \end{cases} \tag{2.39}$$

The algorithm process using gradient descent is shown in Table 2.1 [68]. It is a generic learning algorithm that is based on stochastic gradient descent with the bootstrap sampling of training triplets.

Table 2.1: LearnBPR algorithm

Algorithm 1: LearnBPR(D_S, θ)
Initialize θ
repeat
randomly choose (u,i,j) from D_S
$\hat{\theta} \leftarrow \theta + \alpha \left(\frac{e^{\hat{x}_{uij}}}{1+e^{\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\theta \cdot \theta \right)$
until convergence
return $\hat{\theta}$

where D_S represents dataset, α represents the learning rate and $\hat{\theta}$ is the updated θ.

BPR algorithm outperforms pure matrix factorization algorithm in the application of dealing with implicit feedback [68].

2.2.6.3.2 Weighted Approximate-Rank Pairwise (WARP) Loss WARP is also a pairwise approach and quite similar to BPR [73]. It uses stochastic gradient descent and a novel sampling trick to optimize ranks. This approach is an efficient online optimization strategy. The system calculates the difference between the prediction ratings of a positive and a negative item sampled randomly from a user.

The loss function is :

$$err(f(x), y) = \sum_{i \neq y} L(rank_y(f(x))) \frac{|1 + f_y(x) + f_i(x)|_+}{rank_y(f(x))} \tag{2.40}$$

where $rank_y(f(x))$ is the rank of the true label y given by $f(x)$, $f_y(x)$ can be represent as $f_y(x) = W_y^T H_x$, $|m|_+$ is the positive part of m.

If the system uses the BPR approach, it will make the stochastic gradient descent (SGD) update with this difference as the weight. However, if the system uses the WARP

approach, it will run the SGD update only when the system yields a wrong prediction such as the negative item has a higher predicted score than the positive item. And the system will continue to sample the negative items until it either gets a wrong prediction or reaches the preset value (the max number that the system can sample). The sampling process can be described as follows:

1. Sample a pair (x, y).
2. For the chosen (x, y), sample a violating label $\overline{y}$ such that $1 + f_{\overline{y}}(x) > f_y(x)$.

The algorithm of the optimization of WARP approach is:

Table 2.2: WARP approach optimization

Algorithm 2: WARP approach optimization

Input : labeled data $(x_i, y_i), y_i \in 1, ..., Y$

repeat

randomly choose (x_i, y_i)

$f_{y_i}(x_i) = \Phi w(y_i)\Phi(x_i) = W_y^T V_x$

Set $N = 0$

repeat

randomly choose $\overline{y} \in 1, ..., Y, excludey_i$

let $f_{\overline{y}}(x_i) = \Phi w(\overline{y}_i)^T \Phi(x_i)$

N=N+1

until $N \geq Y - 1$ or $f_{\overline{y}}(x_i) > f_{y_i}(x_i) - 1$

if $f_{\overline{y}}(x_i) > f_{y_i}(x_i) - 1$ then

make a gradient step to minimize:

$L(\lfloor \frac{Y-1}{N} \rfloor)|1 + f_y(x) + f_i(x)|_+$

end if

until validation error does not improve or smaller than a certain value

$\lfloor \quad \rfloor$ means round down.

WARP does introduce two hyper-parameters. One is the margin which determines how wrong your prediction should be when implementing the SGD update. In the paper [73], the margin one means that you must guess $f_{\overline{y}}(x_i) > f_{y_i}(x_i) - 1$ in order to implement the SGD update. The other hyperparameter is the cutoff, which determines how many negative samples you are willing to draw to get a wrong prediction before you give up and move on to the next user.

The WARP can maximize the rank of positive examples by repeatedly sampling negative examples until a wrong prediction is found.

2.2.6.3.3 k-th order statistic loss (WARP-KOS) This is a special edition of WARP that does the pairwise updates based on the k-th positive example for any given user. This method can help the recommender provide more related items in the recommendation list [74]. So, k value represents the positive items we picked in each WARP process.

2.2.7 LightFM Model

LightFM is a python library for both implicit and explicit feedback. It is a hybrid matrix factorization model proposed by Maciej KulaIt [75]. It has three different types of Pairwise approaches (BPR,WARP,WARP-KOS) mentioned in section 2.2.6.3, and two learning rate schedules (Adagrad, Adadelta) mentioned in section 2.2.4. It also makes it possible to incorporate both item and user metadata into the traditional matrix factorization algorithms. It represents each user and item as the sum of the latent representations of their features, thus allowing recommendations to generalise to new items (via item features) and to new users (via user features).

We mention this model here because we are going to use it to run our experiments in Chapter 4.

2.3 Related Work

In restaurant recommender system area, there are a lot of classical papers. Lee and Teng [35] proposed collaborative recommender with multi-criteria decision analysis using a dataset from Zagat. It considers multiple aspects of the restaurant include food, decor, service and price. Zhang et al. [36] proposed a model that uses both implicit (e.g. demographic and geographic information) and explicit (e.g. ratings and reviews) preferences of users to recommend restaurants. For example, a lot of restaurant recommendation websites, such as Yelp, Zagat and Dianping, are using location-based services to both website and mobile users.

Products are usually recommended based on their characteristics. In paper [37], a tourism recommender system uses places that users liked in the past to build user profile and recommends new spots based on it and features of spots. Moreover, recommendations can be altered if users set some constraints, such as time or cost [38, 39, 40]. Experts preset the features of spots, which include available time, geographical information and the information of why this place worth a visit. Especially, in paper [41], researchers generate seasonal feature vectors for spots to represent their characteristics.

Those constraints made by users are very similar to the context element we faced when we deal with restaurant recommender systems. Oku et al. [42] proposed a context-aware support vector machine (SVM) method by using SVM to calculate the similarity between restaurants based on the context parameters of users and characteristics of restaurants. Especially, the SVM method views the set of liked and disliked items of a user in various contexts as two set of vectors in an n-dimensional space. A hyperplane has been constructed to separate this space and maximize the separation between liked and disliked items. In paper [43], contextual eVSM (enhanced Vector Space Model) [44] extends eVSM with a context-aware post-filtering algorithm. More specifically, a semantic representation of the context is built and used to influence non-contextual recommendations. For example, restaurants that have tags such as “seafood” and “Need To Dress” are more relevant if the user is looking for a restaurant that is suitable for a formal dinner. Experiments

demonstrated that contextual eVSM outperformed non-contextual models including the state of the art algorithm for context-aware collaborative recommendation proposed in [45]in most experimental settings.

Tag-based recommenders belong to CBR, which has similar approach to dealing with the datasets like the Entrée dataset. In order to generate recommendations using tags in the dataset, some researchers educed the semantic meaning of tags to calculate the similarity [46, 47, 48]. Moreover, there are a few proposed methods using tags to compute the users' similarity [49, 50, 51]. Researchers in [52] presented a recommender system which made recommendations about websites based on their tag similarity.

In other words, for the purpose of leveraging personalized resources, an extension method is to compute similarity between tags in a way that combines the cosine similarity with other context-aware factors such as the frequency and popularity of tags [53]. Paper [46] proposed a new method which incorporated tags in IBCF algorithm and applied three combinations of two-dimensional correlations between items, tags and users. Tags not only help users to find their interesting items but also can be used to group items and make recommendations [48].

An example of using implicit contextual variables was presented in paper [54]. As another example of contextual information, popular websites such as Yelp and Dianping also consider online reviews. These reviews contain plenty of contextual information describing particular purchases or consumption experiences, such as restaurant visits. For example, the user might indicate in a review that he/she went to the restaurant for dinner with her father to celebrate Fathter's day. Bauman and Tuzhilin [55] presented a method of analyzing such online reviews and extracting contextual information from them.

Our approach also considers the navigation behaviors as some constraints. We map these behavior to scores. Similar to those systems using tags, we also use tags, which represent the characteristics in this system, along with other information to recommend. We use the weighted similarity approach to combining the tag factors and user behavior factors. Moreover, we use a hybrid matrix factorization model, LightFM, to run experiments with some pairwise approaches to learning to rank methods.

2.4 Summary

In this chapter, we presented about the common types of recommender systems and their recent research situations including collaborative-filtering model(UBCF, IBCF), content-based model and context-aware model. Secondly, we showed the process of creating a supervised learning model and trained it with some optimization approaches. Thirdly, we discussed matrix factorization approach and some learn to rank methods that are used in our experiments. Moreover, we provided an overview of restaurant recommendations and their potential applications. Also, the strengths and weaknesses of some well-known methods for using implicit behavior were discussed. In the next chapters, we will present a thorough analysis of data scrubbing process and methodologies to address historical data issues and weighted similarity methods between users and restaurants.

Chapter 3

Methodology

In this chapter, we explore an approach to leveraging user behaviors in section 3.1. Moreover, we propose a data scrubbing process to exploit users' historical data in section 3.2. Finally, we propose a similarity leveraging approach to reveal the real connections between users and restaurants in section 3.3 and 3.4.

3.1 Dataset

The most common form of user behavior data is the log. Websites have produced a significant amount of raw logs during operations. After storing those logs, many Internet businesses will classify different kinds of raw logs into session logs according to user behaviors. Each session represents user behaviors and the corresponding services in one interaction.

In this thesis, we will use the Entrée data set. The website collected the data from their users from September 1996 to April 1999 and organized them according to year quarter (Q). Fig 3.1 shows the data compositions.

Specifically, in Q3 1996 and Q2 1999, each quarter contains only one month. To understand the dataset better, we consider the user behaviors, tags of restaurants, which are depicted as "tag set" in Fig.3.1, users and items have some connections with each

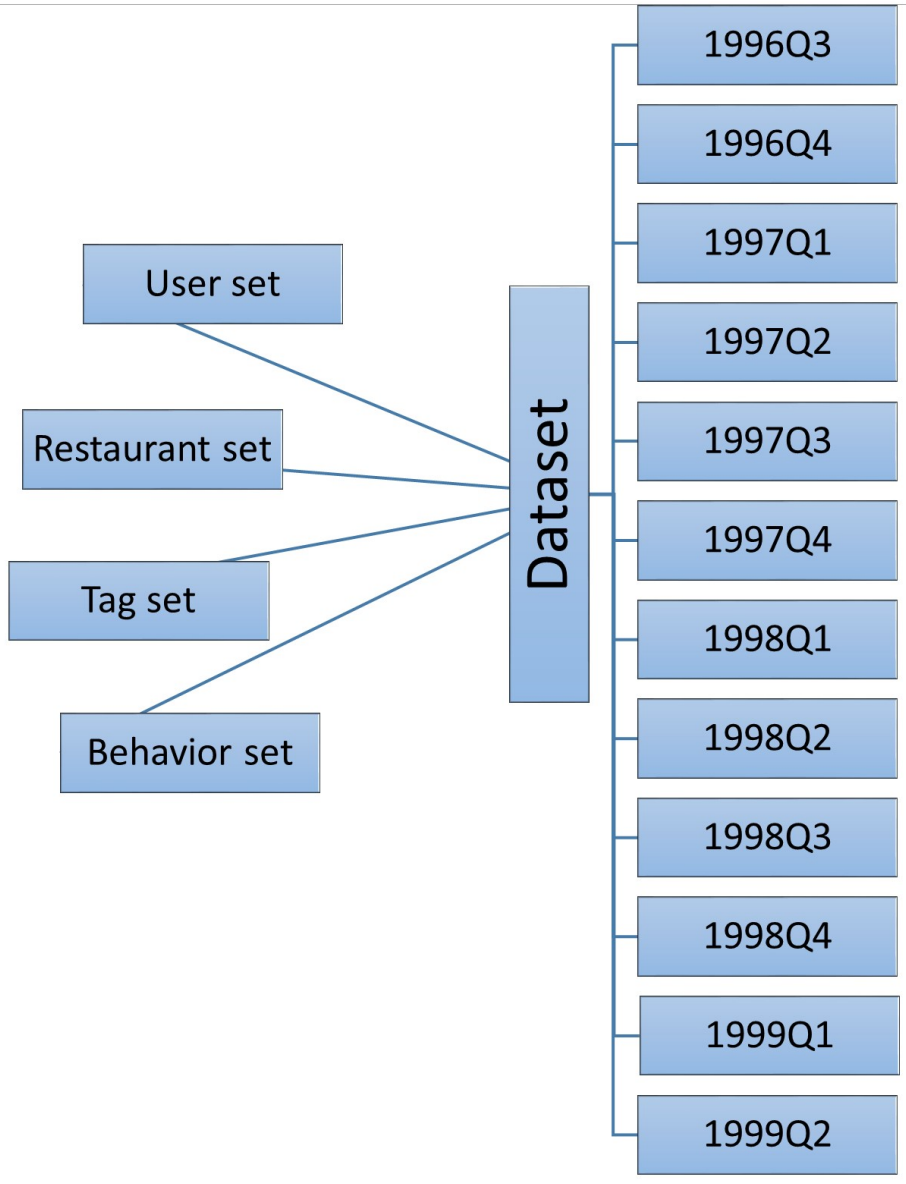

Figure 3.1: Dataset structure.

other. We set $U = \{u_1, u_2, ..., u_n\}$ as the user set, $R = \{r_1, r_2, ..., r_n\}$ as the restaurant set, $T = \{t_1, t_2, ..., t_n\}$ as tag set and $S = \{s_1, s_2, ..., s_n\}$ as context set of user behaviors.

The T set has 256 preseted elements including "Cafeterias", "Asian", "$30-$50", etc. A tag is a word or a phrase that describes the characteristic of restaurants.

The S set has ten actions including both the "Entry point" and "Exit point" and other eight behaviors labeled from L to T shown in Table 3.1, which represent the following information respectively [12][76].

The original dataset documents every click of the users. Each line in the dataset represents a session of a user's interaction with the system. Each line of the dataset is structured as follows:

Date, IP(IP address of user's computer), Entry point, Rated restaurant 1 (label) [tab] restaurant name, ..., Rated restaurant N (label) [tab] restaurant name, Exit point

The dataset has a total of 50672 lines in the above format.

Table 3.1: User behaviors explanations.

Label	Behaviors	Explanation
L	Browse	Jumping from one restaurant to another one.
M	Cheaper	Find a similar but cheaper restaurant.
N	Nicer	Find a similar but higher class restaurant.
P	More traditional	Want to have a more traditional menu.
Q	More creative	Want to have a more creative menu.
R	More lively	Find a more lively restaurant.
S	Quieter	Find a quieter restaurant.
T	Change cuisine	Find a restaurant with an entirely different menu.

In the following subsections, we will transform the context information to some specific ratings[77][28] including both explicit evaluation [76] and implicit evaluation [78].

3.1.1 Explicit Behavior

Explicit behavior's most common representation is rating. The ratings given by users can always show the system their preferences clearly. For example, a user will love an item rated for five points more than a four-point item. Other than ratings, there are more kinds of explicit feedback such as buying history and like or thumbs up button in the videos or music.

In our dataset, the first part of explicit behavior is the content that represents the positive feedback from users. In the Entrée system, the Entry Point is entered by the user, then the system will recommend some restaurants similar to the Entry Point restaurant, according to the characteristics of this restaurant. It is very likely that the user will choose their most ideal restaurant as the standard to find similar candidates. Therefore, for this kind of restaurants, we use a function $R(u, r) = 1$ to express the strongest interest rate.

The second part is the behaviors that represent the negative feedback from users. For the navigation choices such as "Nicer", "More Traditional", "More Creative", "Cheaper", "More Lively", "Quieter" and "Change Cuisine", which indicate that the users do not like the restaurant, and the restaurant will not be their ideal candidate at least for a certain period. Therefore, we score them as $R(u, r) = -1$, which represents a strong dislike to the restaurant.

3.1.2 Implicit Behavior

The opposite to explicit behavior is implicit behavior, which means the behavior may not represent the user preferences clearly. The most common representation of implicit behavior is web-browsing. User browsing a page may not indicate that the user loves the item of information on this page. It is possible that the link to this page displays on the home page where user is more likely to click, or it is hidden under an advertisement where

the user just mis-click it. As we can see in the above paragraph, the explicit feedback can be classified to positive part or negative part easily while this is harder to be done for implicit behaviors.

For "Exit point", the user would confirm the end of the search on this "Exit point" (for some users that directly log out without confirming the "Exit point", the system will not record this value). If there is an "Exit point", we consider that the user is very likely to choose this restaurant. But this is not for sure because they may just follow the ending process. Therefore, for such implicit behavior, we can only give $R(u, r) = 0.8$, assuming that the user is very likely to choose this kind of restaurant.

For another behavior, "Browse", it means users do not like this restaurant while they do not give the restaurant a negative evaluation directly and do not seem to choose this kind of restaurants eventually. There is no critique like "Nicer" or "Cheaper" for the restaurant because either users do not know why they dislike this restaurant or their reasons of dislike are not listed in the website. In this case, "browse" is relatively a mild negative implicit behavior. Therefore, we give $R(u, r) = -0.5$.

We summarize these points in Table 3.2 below.

Table 3.2: The mapping approach of user behaviors to scores

Behaviors	Scores
Browse	-0.5
Change Cuisine	-1
Cheaper	-1
Nicer	-1
More Creative	-1
More Traditional	-1
More Lively	-1
Quieter	-1
Entry Point	1
Exit Point	0.8

After finishing the mapping approach from behaviors to scores, we have Table 3.3.

Table 3.3: Original Dataset

User	Restaurant	Positive Rating	Negative Rating
50672	1149	50583	189291

At this time, the density of the dataset, which is calculated by the fraction of filled ratings to total ratings, is 0.412%. It is a very sparse dataset. In practice, datasets as sparse as this one are hard for the recommender system to utilize those data and provide proper recommendation because the similarities between users or items are too small. So we need to use some mechanisms to make this matrix become less sparse. In the next

section (section 3.2), we will introduce the pre-processing approach that we proposed to solve the sparsity issue and leverage users' historical data.

3.2 Data Scrubbing Process

The dataset includes over 50000 visits information (original dataset treats every visit as a new user), many of which are too sparse; for example, some users browsed very quickly and departed directly in a few minutes. Therefore, we proposed a data scrubbing process consisting of the following steps to make the dataset less sparse and leverage the usesr' historical ratings:

1. Users may use the system many times in a short period. In this short period, for example in a quarter, users are likely to keep the same dining preference such as preferring spicy food in winter or prefer light or diet food in summer. And it is common that users have different preferences in different seasons. So, we consider the same IP address represents the same user in the same quarter of the year. In this case, the number of users will decrease to almost twenty-two thousand. [1]

2. Since the Entrée system allows users to use a restaurant in other cities within the database as the "Entry point" (but only recommend the restaurant locate in Chicago), there are only 676 restaurants in the database of Chicago (the others are not located in Chicago). Therefore, we will remove those restaurants outside Chicago.

3. We remove users that do not have any positive feedback, at least either one "Entry point" or "Exit point". Since we remove some restaurants from other cities, some users may have no "Entry point".

4. The number of restaurants evaluated by a user must be more than 15 so that we can have a less sparse rating matrix, and make sure that every user in the rating matrix

[1] In the Entrée dataset, IP address is the only data we can utilize to address the user profile problem. In the modern society, when we design a new recommender system, we should allow users to create their own accounts rather than use the IP address only to identify users.

has enough information for recommender systems. Because the Entrée dataset was documented as log, length of each visit in the Entrée dataset range from one to twenty interactions. According to paper [79], 10 or 15 examples would not be enough to provide good recommendations for any one user, we set our threshold to 15.

5. To avoid the issue of sparsity, which may result in zero value similarity and harm the recommendation quality, the restaurants that have less than three ratings should be removed.

6. Users may rate the same restaurant many times. We calculate the sum of all the ratings that each user gives to the same restaurant. Therefore, we can utilize all the information on how a user thinks about a certain restaurant. We set the rating to 1 if the calculated sum is larger than 1 and set the rating to -1 if the calculated sum is smaller than -1.

   For example, user A has rated restaurant A for 3 times in different recorded sessions, such as (1, Entree point), (-0.5, browse), and (0.8, Exit point). The sum of all three scores is 1.3, which is larger than 1, then we set 1 as the score rated from user A to restaurant A. Likewise, if the sum is smaller than -1, we will set -1 as the score.

   In this way, we make use of every single rating of the users to get a better prediction.

After the preprocessing step, the information of our dataset is shown in Table 3.4. We retain 672 out of 676 restaurants in Chicago. In the mean time, we retain 4089 users because we treat visits with the same IP address as belonging to a single user. Since a lot of restaurants have more than one rating per visit from the same IP addresses, also a quite amount of them input a restaurant outside Chicago as "Entry point" and then finish the visit without an "Exit point". In this case, some visits do not contain a positive rating, which means these visits are meaningless if we treat each one of them represent a unique user. Although there are only eighty-five thousand ratings retained after the preprocessing step, those eighty-five thousand ratings can still represent most of the meaningful part of the dataset.

Besides, the density of the dataset increased to 2.92%, which is a normal density rate in recommender system area. Considering these factors, our data scrubbing process is necessary in order to avoid the sparsity problem, and would not change core of this dataset.

Table 3.4: Dataset after pre-processing

User	Restaurant	Positive Rating	Negative Rating
4089	672	15975	64245

3.3 Context-Aware Model based on User Behaviors

In this section, we are going to present our work on leveraging those user behaviors, which utilizes a heuristic approach to the opposite behaviors and tag feature, to make use of the massive and sparse data.

3.3.1 Learning Sparse Context of User Behaviors

The subjectiveness of a user is indicated by the behavior buttons that the user has pressed, such as Nicer and quieter, etc. Although the mapping of user behaviors to scores is good for matrix factorization and collaborative filtering algorithms, it does not carry any information about the differences in different behaviors. We will lose a lot of information regarding users' subjectiveness if we only use the scores we give to different behaviors. For example, if user A is browsing restaurant b and going to click the button "cheaper" while user B is browsing the same restaurant but thinking it is not quiet enough, both users will be considered rating -1 to this restaurant, which means the system will treat them as users with similar preference. In this case, user A and B share no similarity in this restaurant b in reality, but they might get the same recommendation by the system in the next web page.

To avoid the loss of substantial context information, we can build a three-dimensional context-aware model as illustrated in Fig.3.2.

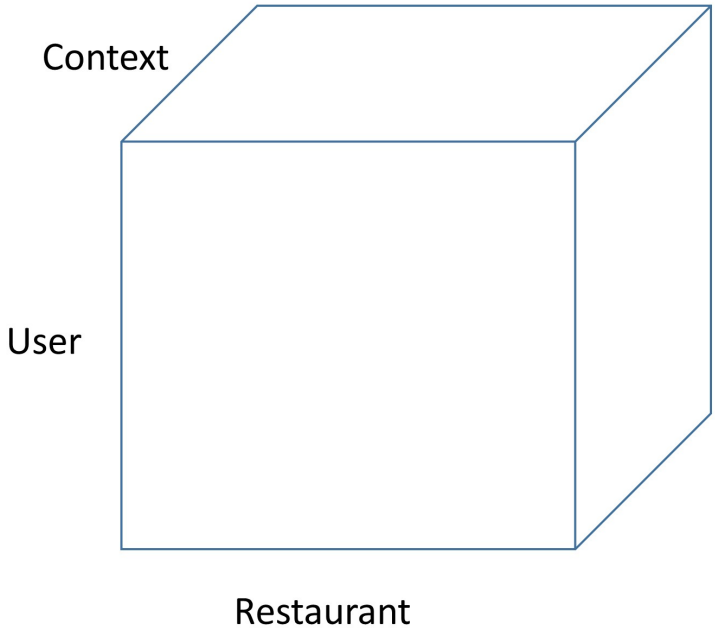

Figure 3.2: Three-dimensional context-aware model.

This model considers the context only, and uses these ten behaviors to calculate the similarity. We transform each element in matrix $R(u, r)$ into a vector V, where V is a ten-dimensional vector (which is Browse, Cheaper, Nicer, More traditional, More creative, More lively, Quieter, Change cuisine, Entry point, Exit point). Let v_k be the k-th component of V. Then v_k will be set to 1 when user u clicks one of the navigation buttons of restaurant r or set it as the "Entry point" or "Exit point" on the web page; otherwise, v_k will be set to 0, as described in equation (3.1):

$$R(u, r, v) = \begin{cases} 1, & u \quad critiques \quad r \quad at \quad v_k \\ 0, & otherwise \end{cases} \tag{3.1}$$

In this case, both user A and B can critique the same restaurant b in different dimensions on the vector V, which can be called the context vector. Therefore, we can calculate the similarities between users using the behaviors' similarity matrix as in Table 3.5. In this table, we consider each behavior is independent and have no relationship with other behaviors.

Table 3.5: Sparse similarity matrix for Entée behaviors

Br.	Ch.	Ni.	Tr.	Cr.	Li.	Qu.	Cu.	En.	Ex	
1	0	0	0	0	0	0	0	0	0	Browse
	1	0	0	0	0	0	0	0	0	Cheaper
		1	0	0	0	0	0	0	0	Nicer
			1	0	0	0	0	0	0	Trad.
				1	0	0	0	0	0	Creat.
					1	0	0	0	0	More Lively
						1	0	0	0	Quieter
							1	0	0	Cuisine
								1	0	Entry
									1	Exit

Using this table, we can calculate restaurants' similarities using the cosine-based approach as equation (3.2):

$$sparse\_sim(A, B) = \frac{\vec{V_A} \cdot \vec{V_B}}{|\vec{V_A}| * |\vec{V_B}|}, \tag{3.2}$$

where $sparse$ means this similarity calculation is using a relatively sparse behaviors' similarity matrix than the heuristic approach we are going to present in the next subsection, $sim(A, B)$ represents similarity between users A and B, $\vec{V_A}$ is the context vector of user A while $\vec{V_B}$ is the context vector of user B. More details of the calculation process will be

given in section 3.3.1.1.

3.3.1.1 Example

If we have two users logs as shown in this table:

Table 3.6: Example data

	A	B	C
Amy	-0.5(Browse)	1(Entry point)	-1(Nicer)
David	0	0.8(Exit point)	-1(Nicer)

In this table, the logs are written in "score(behavior)" format, and A, B, and C are the restaurants rated by either Amy or David. Then we can extract the context vectors of these two users as shown in the following table:

Table 3.7: Context vector of Amy and David

		Amy	David
A	Browse	1	0
B	Entry point	1	0
	Exit point	0	1
C	Nicer	1	1

In Table 3.7, we have two context vector where Amy is $(1, 1, 0, 1)$ and David is $(0, 0, 1, 1)$. The similarity between user Amy and David can be calculated using equation (3.2) as follow:

$$Sim(Amy, David) = \frac{(1,1,0,1) \cdot (0,0,1,1)}{\left|\sqrt{1^2+1^2+0^2+1^2}\right| * \left|\sqrt{0^2+0^2+1^2+1^2}\right|} = 0.408 \qquad (3.3)$$

As we can see in the equation (3.3), ratings are not used in this calculation.

3.3.2 Heuristic Similarity Approach to Learn User Behaviors

If we use the approach above to store the users' ratings, our dataset would have thousands of ten-dimensional vectors data. It will be very sparse in such a high-dimensional space which will make the similarities between users close to each other and also close to zero. In that case, it will be hard for the recommender system to reach a good result.

Therefore, we propose a new heuristic similarity calculation approach based on the method from Entrée C system [79].

1. We take the scores in Table 3.2 into consideration along with ten behaviors. So each restaurant's evaluation will become a action pair including rating and behavior.

2. Some behaviors represent opposite meanings, such as "Nicer"/"Cheaper", "Quieter"/ "More Lively", and "More Creative"/"More Traditional". We consider they can be stored oppositely in the context dimension using 1 for the behavior before the slash and -1 for the latter one. In this case, we can group these opposite behaviors into pairs in our heuristic similarity matrix.

3. In Entrée C system, it assumes that some behaviors have semi-opposite behavior pairs such as "Nicer" and "More Lively" along with semi-similar behavior pairs such as "More Traditional" and "Nicer". For example, both a "More Lively" restaurant and a "More Traditional" restaurant do not has to be Nicer, they can be cheaper than the former choice. If we use these relations, we will add noise into the database. Therefore, these behavior pairs with "half connection" will be removed.

4. We find out that the 'change cuisine' behavior never exist in our dataset. Alternatively, we can say that no user would like to use this kind of button to change the whole menu while they are using the website. The reason may be when users can not find the restaurant they like they would choose exit directly instead of clicking 'change cuisine'. So we will remove this behavior from the context.

After these adjustments, we group the relative behaviors together as shown in Table 3.8. Using this heuristic approach, we manage to reduce the dimensionality of context from ten

to four. In this table, "Br." represents "Browse","En." represents "Entry point", "Ex." represents "Exit point", "Ni." represents "Nicer", "Ch." represents "Cheaper", "Qu." represents "Quieter", "Li." represents "More Lively", "Cr." represents "More Creative", and "Tr." represents "More Traditional"

Table 3.8: Heuristic Similarity matrix for Entée behavior groups

En./Ex./Br.	Ch./Ni.	Tr./Cr.	Li./Qu.	
1	-1	-1	-1	Entry point/ Exit point/Browse
	1	0	0	Cheaper/ Nicer
		1	0	More Traditional/ More Creative
			1	More Lively/Quieter

In the calculation of heuristic similarity, we use the average distance of common action pairs between different users to represent their similarities, which include the rating and the behavior that the user gives to a certain restaurant. Let the distance function between the common action pairs be $dis(a,b) = \vec{H}_a \cdot \vec{H}_b$, where a and b represents the rating and the behavior given by users A and B to a certain restaurant respectively and vector $\vec{H}$ represents the action pair of a certain restaurant. Then we have the average distance between two users can be calculated as:

$$heuristic\_sim(A,B) = \frac{\sum_{r \in R} dis(a,b)}{|R|}, \tag{3.4}$$

where *heuristic* means this similarity calculation is using a heuristic way calculate the similarities between restaurants, sim(A,B) represents similarity between user A and B, R represents the co-rated restaurants set, $|R|$ represents the number of element in R, and r represents a restaurant in the R set. More details of the calculation process will be given in section 3.3.2.1.

3.3.2.1 Example

We can use the data in Table 3.6 to show the difference of calculations between *sparse_sim* and *heuristic_sim*. We can extract the action pairs of Amy and David from this table.

Table 3.9: Action pairs of Amy and David

		Amy	David
A	rating	-0.5	0
	Browse	1	0
B	rating	1	0.8
	Entry point	1	0
	Exit point	0	1
C	rating	-1	-1
	Nicer	1	1

As shown in Table 3.9, Amy and David has common action pairs in restaurant B and C. The similarity between Amy and David can be calculated as:

$$heuristic\_sim(Amy, David) = \frac{(1,1,0)\cdot(0.8,0,1)+(-1,1)\cdot(-1,1)}{2} = 1.4 \qquad (3.5)$$

3.3.3 Comparison of Similarity Methods

We will compare the similarity methods in an example that contains four different users in the Entrée dataset to see whether the heuristic technique is better than the learning sparse context or the rating vector approach.

Table 3.10 shows the logs we extract from the dataset after mapping their behaviors to score and writes them into "score(behavior)" format in the table.

Table 3.10: Entrée logs in tabular form

	A	B	C	D	E	F
Amy	-0.5(Br.)	1(En.)	-1(Ni.)	-1(Ni.)	-1(Cr.)	-1(Ni.)
Betty	-1(Li.)	0	-1(Ch.)	-1(Ch.)	-1(Tr.)	-1(Ch.)
Candy	0	0	-1(Ni.)	0.8(Ex.)	1(En.)	-1(Ch.)
David	0	0.8(Ex.)	-1(Ni.)	-1(Qu.)	0	0

We will calculate the similarity using Pearson correlation function (equation 2.1) between Amy and three other people. The results are report in Table 3.11. From Table 3.11, we can see Betty and Amy share almost the same preference. Both of them gave negative ratings to restaurant A, C, D, E and F, although there is a little difference in their rating to restaurant A. For restaurant B, Amy gave a positive rating while Betty did not rate it. But this will not be considered in the Pearson correlation function.

Table 3.11: Correlation comparison based on rating vector

User	Similarity to Amy
Betty	0.97
Candy	0.02
David	0.76

In fact, Amy and Betty have opposite behaviors on restaurant C, D, F. Amy thought they are not luxurious enough while Betty thought they are too expensive. These opposite behaviors suggest Amy and Betty may belong to different consumer groups and may indicate they have different level of income. It would be inappropriate to recommend the restaurants Amy likes to Betty because they would be too expensive to Betty.

Next, we will calculate their similarities using sparse metric. We transformed the three-dimensional data into two dimensions to provide a good readability as Table 3.12. We can use equation (3.2) to calculate the similarity. The results are shown in Table 3.13.

Table 3.12: Sparse metric representation

		Amy	Betty	Candy	David
A	Br.	1	0	0	0
	Li.	0	1	0	0
B	En.	1	0	0	0
	Ex	0	0	0	1
C	Ni.	1	0	1	1
	Ch.	0	1	0	0
D	Ni.	1	0	0	0
	Ch.	0	1	0	0
	Ex.	0	0	1	0
	Qu.	0	0	0	1
E	Cr.	1	0	0	0
	Tr.	0	1	0	0
	En.	0	0	1	0
F	Ni.	1	0	0	0
	Ch.	0	1	1	0

Table 3.13: Correlation comparison based on sparse metric

User	Similarity to Amy
Betty	0
Candy	0.41
David	0.24

The results show that Betty is the most irrelevant person to Amy, which supports our analysis above. Since Candy becomes the most relevant person to Amy, let us look into their behaviors. They share the same preference on restaurant C as they both click the button "Nicer" to find a nicer restaurant. On the other hand, they are divided in opinion on restaurant D, E and F. In this case, Candy does not seem to be the correct relevant person.

Finally, let us take a look at the heuristic approach. As we did for Table 3.12, we also transform the data into two-dimension, as shown in Table 3.14. We can see the difference between using the opposite behavior pair as the context vector and using a single context vector as in Table 3.12.

Table 3.14: Heuristic metric representation

		Amy	Betty	Candy	David
A	Rating	-0.5	-1	0	0
	Br.	1	0	0	0
	Li.	0	1	0	0
B	Rating	1	0	0	0.8
	En.	1	0	0	0
	Ex.	0	0	0	1
C	Rating	-1	-1	-1	-1
	Ni./Ch.	1	-1	1	1
D	Rating	-1	-1	0.8	-1
	Ex.	0	0	1	0
	Ni./Ch.	1	-1	0	0
	Qu./Li.	0	0	0	1
E	Rating	-1	-1	1	0
	En.	0	0	1	0
	Cr./Tr.	1	-1	0	0
F	Rating	-1	-1	-1	0
	Ni./Ch.	1	-1	-1	0

Also, we use equation (3.4) to calculate the similarities. Results are listed in Table 3.15, where each row represents the similarity between other users and Amy, and the final row is the average similarity. A Missing cell represents either one of them did not rate this restaurant. It shows that David is the most relevant person to Amy at this time. Although

there are only three restaurants that Amy and David co-rated, they share the same like and dislikes feelings on all three restaurants. Moreover, there is no opposite behavior in the logs. These two reasons allow David to achieve an extremely high score in the calculation of similarities.

Table 3.15: Correlation comparison based on heuristic metric.

	Betty	Candy	David
A	0.5		
B			0.8
C	0	2	2
D	0	-0.8	1
E	0	-1	
F	0	0	
Average Sim.	0.1	0.08	1.26

After comparing all three models, we find that the heuristic metric is much closer to the real semantics of user behaviors. Also, it is better than the sparse metric with a smaller storage requirement and faster processing time because we reduce the number of elements in the context vector from ten to four.

3.4 Improvement on Restaurant Similarities

In the previous section, we have discussed our development of the user's heuristic similarity, from cosine-based correlation and Pearson correlation to sparse similarity and heuristic similarity approaches for learning use behaviors. In fact, we can also use this kind of process to calculate the similarity between restaurants. In the following subsection, we will present it briefly because the steps of this process are quite similar to those in the previous section.

3.4.1 Heuristic Similarity for Restaurants

When we are considering restaurant's similarity, we think that every user has the preference on ratings. Some users prefer to use button "Cheaper" to navigate the system because their incomes are low; some old users prefer to use button 'More Traditional' because they prefer old style restaurants. In order to reduce this preference's effect, we suppose users' preferences are represented as the average ratings of restaurants. The rating preference of the single user can be represented as follows:

$$prefer(u) = \frac{\sum_{k \in K} R_{u_k}}{|K|} \tag{3.6}$$

where K represents the whole set of rated restaurants from user u, R_{u_k} represents the rating given from user u to restaurant k. So we have the preference vector for all users as: $\bar{U} = \quad (prefer(u_1), \quad prefer(u_2), \quad ..., \quad prefer(u_n))^T$.

The similarities between restaurants can be calculated in a similar way to how we do with users. The equation looks like this if we add user preferences into consideration:

$$hr\_sim(i,j) = \frac{\sum_{i \in L} (V^i - \bar{U} \cdot I_i) \cdot (V^j - \bar{U} \cdot I_j)}{|L|} \tag{3.7}$$

where V^k represent the entire rating matrix of restaurant k, I_k is the diagonal matrix contains the non-zero information of V^k and L is the co-rated user set to restaurant i and j.

3.4.2 Combinatorial Similarity with Tags

In the same time, we should pay attention to the tag set, where each tag is a word or a phrase, as we have mentioned in section 3.1. These tags are preseted objective descriptions and representations of restaurants' characteristics. We use $E(r_j, t_k) \in \{0, 1\}$ to represent the relationship of restaurants and tags as equation (3.8).

$$E(r_j, t_k) = \begin{cases} 1, & if \quad restaurant \quad r_j \quad has \quad the \quad tag \quad t_k \\ 0, & otherwise \end{cases}, k \in [1, 256] \tag{3.8}$$

From equation (3.8), we can define the tag vector of restaurant j as $\vec{r_j} = (t_1, t_2, ..., t_k), k = 256$. For example, if restaurant r_j has tag No.4, No.125, No.177 and No.206, its tag vector would have value of one in element t_4, t_{125}, t_{177} and t_{206} while the other 252 elements are zero. The Entrée system used the tag vector of each restaurant to analyze similarity between restaurants and provide recommendations for users. The equation is :

$$tr\_sim(i,j) = \frac{\vec{i} \cdot \vec{j}}{\left\|\vec{i}\right\| \left\|\vec{j}\right\|} \tag{3.9}$$

where $sim(i,j)$ is the similarity measure between restaurant i and j, tr represents tags of restaurants, $\vec{i}$ and $\vec{j}$ are the tag vectors of restaurant i and j. We want to enhance our heuristic similarity with this tag system, which will take advantage of what we have in our dataset to get closer to the real situation. So we proposed the following weighted similarity approach:

$$resta\_sim(i,j) = w_1 \times hr\_sim(i,j) + w_2 \times tr\_sim(i,j) \tag{3.10}$$

where hr_sim represents heuristic restaurant similarity, tr_sim represents tag restaurant similarity, $resta\_sim$ represents weighted sum similarity between hr_sim and tr_sim.

In the next Chapter, experiments will be conducted to select the appropriate weight w_1 and w_2. The general idea is that, we should increase the weight of tr_sim to make $resta\_sim$ more compatible to the technique that Entrée system used. Because tr_sim calculates the similarity between restaurants while the Entrée system makes recommendation using the similarity between restaurants too. If we put less weight in tr_sim, which would make $resta\_sim$ deviate a lot from the original dataset (original similarity), we may encounter a performance drop in the experiment.

3.5 Summary

In this chapter, we presented about the main algorithms we are going to test in Chapter 4 include BPR, WARP and WARP-KOS in LightFM python library. Then, we introduce

what we have in our dataset and how we plan to exploit it.

Moreover, we analyzed the semantics of user behaviors in the Entrée system and projected these behaviors into ratings. Secondly, we take advantage of the meaning of these behaviors in practice and propose an approach to leveraging them. Thirdly, we compare three similarity approaches. Finally, we proposed a new restaurant similarity function by adding the tag similarity of restaurants into the heuristic similarity.

Chapter 4

Evaluation

In this section, we are going to use five different measurements to evaluate the performances of three pairwise LTR algorithms and five classical recommender systems on the Entrée dataset. They are precision, recall@k, F-Score, the AUC value, the MAP value and the MRR value. All these measurements are common in the area of recommender systems, and all of them are focusing on the rank of the recommended items except the AUC value.

The Entrée recommender system aims to provide a list of restaurants as short as possible that meet users' needs. In this case, measurements that focus on ranking are necessary. We will introduce them in the following subsections.

In the experiments, we will run the four-fold cross validation. The dataset is further partitioned into four folds of training and testing sets to be used for cross validation. Each test set has about 20000 ratings. This process is illustrated in Fig.4.1. We use the user and restaurant similarity matrices as their features.

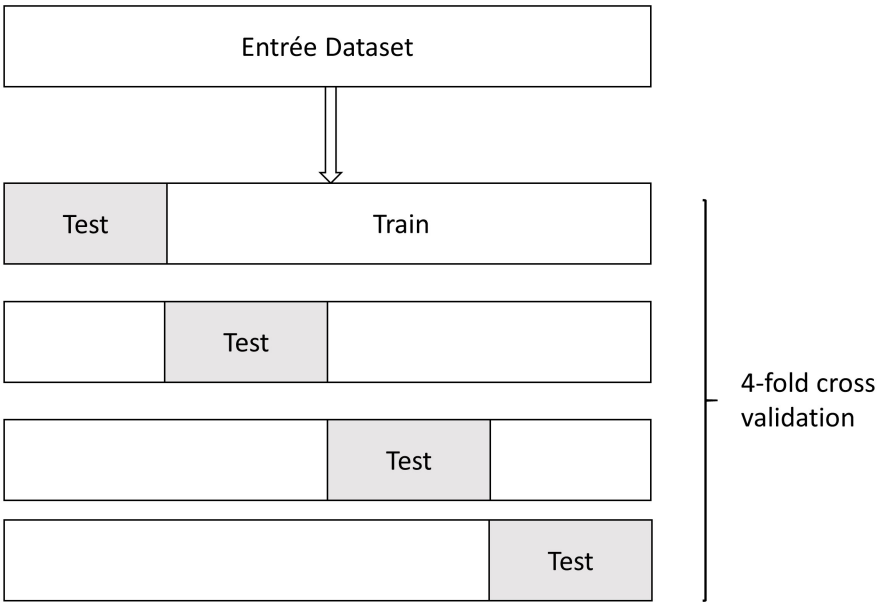

Figure 4.1: The structure of four-fold cross validation

We will compare the results of the LightFM model between three different pairwise approaches namely BPR, WARP and K-OS WARP, and compare the best LightFM performance to five other classic models which are user-based collaborative filtering (UBCF), content-based recommender (CBR), sparse behavior collaborative filtering (SBCF), original heuristic behavior collaborative filtering (OHBCF) and singular value decomposition plus plus (SVD++).

We provide below a brief description of five classic models we used for comparisons and the reasons we choose them in our experiments.

1. UBCF. It uses the Pearson correlation to predict users' preferred restaurants based on the rating only. It is a basic recommendation algorithm that both SBCF and the HBCF are its extensions.

2. CBR. It uses the tags of restaurants as the content that users need to provide recommendations. This method is very similar to the original Entrée system, which uses the navigation as the condition to search the proper restaurants in the database based on the characteristics of restaurants. As we can not run the original Entrée system, it can only be re-run if we go through all those visit in sequence. And this is not very practical in offline experiments. Therefore we use CBR to approximate the original Entrée system's performance.

3. SBCF. It uses the sparse similarity, as shown in Table 3.12, and equation (3.2) to calculate the similarities. This method only considers the similarity between users, because we want to compare the performance of using ratings only in UBCF and using behavior context only in SBCF.

4. OHBCF. It uses the heuristic similarity calculation process in paper [79] to represent the performance of Entrée C system. This method only considers the similarity between users, because we want to compare the performance between the models of using ratings only, using behavior context only, using original heuristic similarity only and using the weighted similarity in our proposed approaches.

5. SVD++. It uses both the users and restaurants feature matrices and stochastic gradient descent approach in training to update both matrices. Then it uses the dot product of user feature matrix and restaurant feature matrix to provide recommendations. [80]. This can be categorized as matrix factorization method. We choose this algorithm because the LightFM model is a hybrid matrix factorization method and we want to compare the differences in the condition of with or without the pairwise approaches.

4.1 Evaluation Metrics

In this thesis, we made some improvements for the Entrée system. The Entrée system receives several negative ratings after the entry point and tries to provide a restaurant that

a user may rate it positively. So, we would like to follow this kind of rules and try to distinguish the positive item from large volume of negative items.

For the specialty of the dataset we are using, we can not use the original top-k method to calculate the precision because:

1. The ratio between the amount of positive and the negative items is a little bit less than a fifth, which means that even if the quality of the recommendation is outstanding, the precision still can not exceed 20%. In the experiment, if the value of evaluation result is too small, the comparison will be insignificant.

2. Our dataset is a sparse dataset with less than four positive feedback per user. If we are running a four-fold cross validation, each test set would have only one positive feedback per user which makes the top-k evaluation meaningless.

Because the prediction accuracy of positive and negative samples in the test set has a positive correlation with the original top-k accuracy. And this new evaluation metric addresses those two problems above. So, we proposed this specific evaluation metrics firstly.

Below are the description of our evaluation metrics.

1. Precision. This metric focuses on calculating the prediction accuracy of positive and negative samples. The precision can be calculated using:

$$precision = \frac{1}{u} \sum_{u} \frac{N_{correct}}{N_{kp}} \tag{4.1}$$

   where N_{kp} represents the number of each user's positively-rated restaurants in the test set, u represent the number of users and $N_{correct}$ represents the number of each user's correct predictions in the test set. Its perfect score is 1.0.

2. Positive recommendations rank first (PRRF). This metric measures the percentage of the correct first recommendation. It can be calculated as:

$$PRRF = \frac{1}{u} \sum_{u} N_{PAF} \tag{4.2}$$

where N_{PAF} represents the number of the items in the first rank is positive.

3. Recall@k. This metric measures the recall at k metric for a model: the number of positive items in the first k positions of the ranked list of results divided by the number of positive items in the test period. The recall@k can be calculated using:

$$Recall@k = \frac{1}{u} \sum_{u} \frac{N_{kp} \cap N_{relate}}{N_{kp}} \tag{4.3}$$

where N_{relate} represents top-k restaurants in each user's test set. Its perfect score is 1.0.

4. F-Score. This metric measures the weighted average of precision and recall. We can pick a parameter λ based on how we balance the value of precision or recall. In this thesis, we pick the number 2 for this parameter as we value them equally.

$$F - score = \frac{\lambda * (precision + recall@k)}{precision * recall@k} \tag{4.4}$$

5. Area Under Curve (AUC). This metric computes the area under the receiver operating characteristic curve (ROC-curve). AUC metric for a model computes the probability that a randomly chosen positive example has a higher score than a randomly chosen negative example.

$$AUC = \frac{1}{|V_{test}|} \sum_{u} \frac{1}{(r_a, r_b) \in V_{test}} P(\hat{y_{u,r_a}} > \hat{y_{u,r_b}}) \tag{4.5}$$

Where V_{test} represents the test set, $|V_{test}|$ represents the number the test set. a and b represent the positive example and the negative example whilst $P(\hat{y_{u,r_a}} > \hat{y_{u,r_b}})$ represents the probability of the restaurant $a's$ prediction score higher than restaurant b. Its perfect score is 1.0.

6. Mean Reciprocal Rank (MRR). This measures the reciprocal rank metric for a model: 1 / the rank of the highest ranked positive example.

$$MRR = \frac{1}{ui} \sum_{u} \sum_{i} \frac{1}{rank_i} \tag{4.6}$$

Where i represent the number of restaurants and $rank_i$ represents the rank of restaurant i in the test set of user u.

7. Mean Average Precision (MAP). This metric measures the mean value of the average precision of each user based on the assumption that the user is interested in finding many relevant restaurants in the system. Average precision represents the average of the precision values from the rank positions where an acceptable restaurant was found.

$$MAP = \frac{1}{u} \sum_{u} \frac{1}{p} \sum_{p} \frac{rank_p}{rank_{t_p}} \tag{4.7}$$

where p represent the set of positively rated restaurants in the test set, $rank_{t_p}$ means the rank of restaurant p in the test set and $rank_p$ means the rank of restaurant p in the positively rated restaurants. Its perfect score is 1. MAP is a very popular evaluation methods because:

A single value measure based on the ranking of all the relevant items.

And the value depends heavily on the highly ranked relevant items. Therefore, the top ranked documents are the most important.

4.2 Experiments

In the LightFM model, we need to optimize a few parameters including learning rate schedule, learning rate, the number of components and the parameter w_1 and w_2 in calculating $resta\_sim$. As AUC value calculates the probability that the prediction score of randomly-picked positive example is have than that of randomly-picked negative one, in the experiment we use the AUC value as the metric to choose different parameters. All the following hyper-parameters are chosen via validation set.

1. Learning rate schedule. In order to find the better learning rate schedule for the Entre dataset, we use the same parameters (learning rate 0.1 for Adagrad only, component number 60, both W_1 and W_2 equals to 0.5, and use WARP-KOS as loss function with K set to default value as 1) to run these two different schedules and the results are shown in Fig. 4.2.

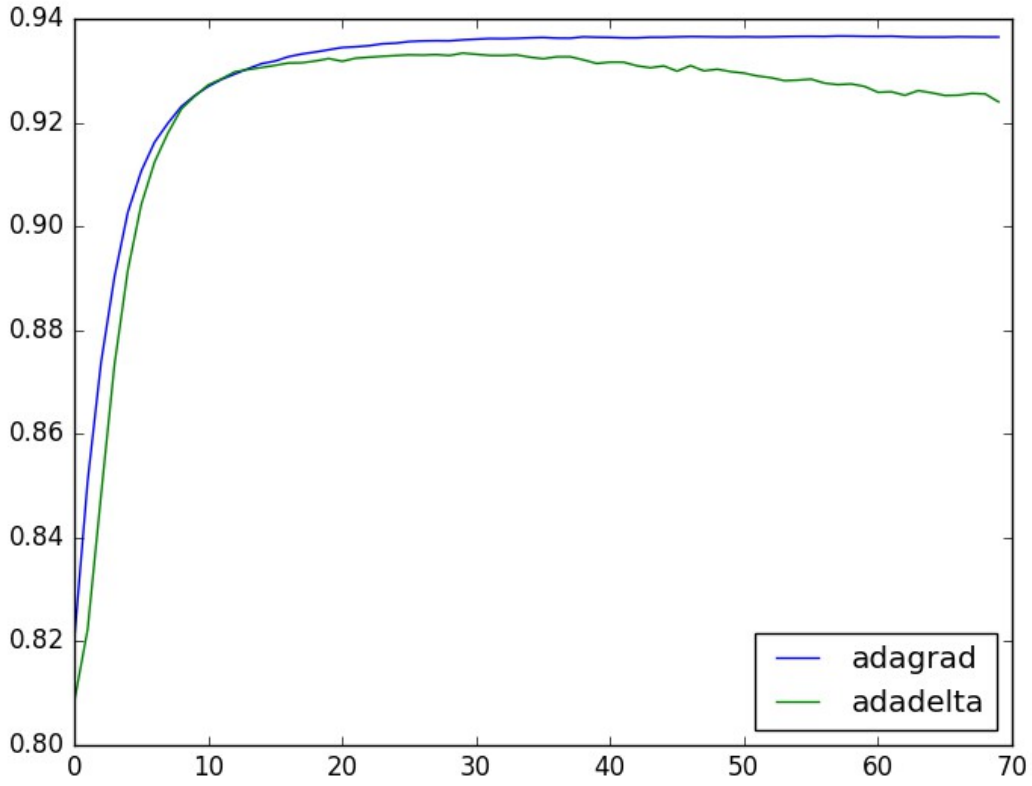

Figure 4.2: AUC for two learning rate schedules.

In Fig. 4.2, the X-axis represents the number of epoches and Y-axis represents the AUC value. As we can see in this figure, we should use the adagrad learning rate schedule as it reaches a higher AUC value with more epoches.

2. Learning rate. The learning schedule we choose is Adagrad, which needs an initial learning rate. We continue to use 60 as the component number and the same loss function as above. Setting learning rate for SGD is usually a process of starting with

a value range from 0.1 to 0.01. Generally, people optimize the model with a large learning rate (0.1 or so), and then reduce this rate until they find a good one.

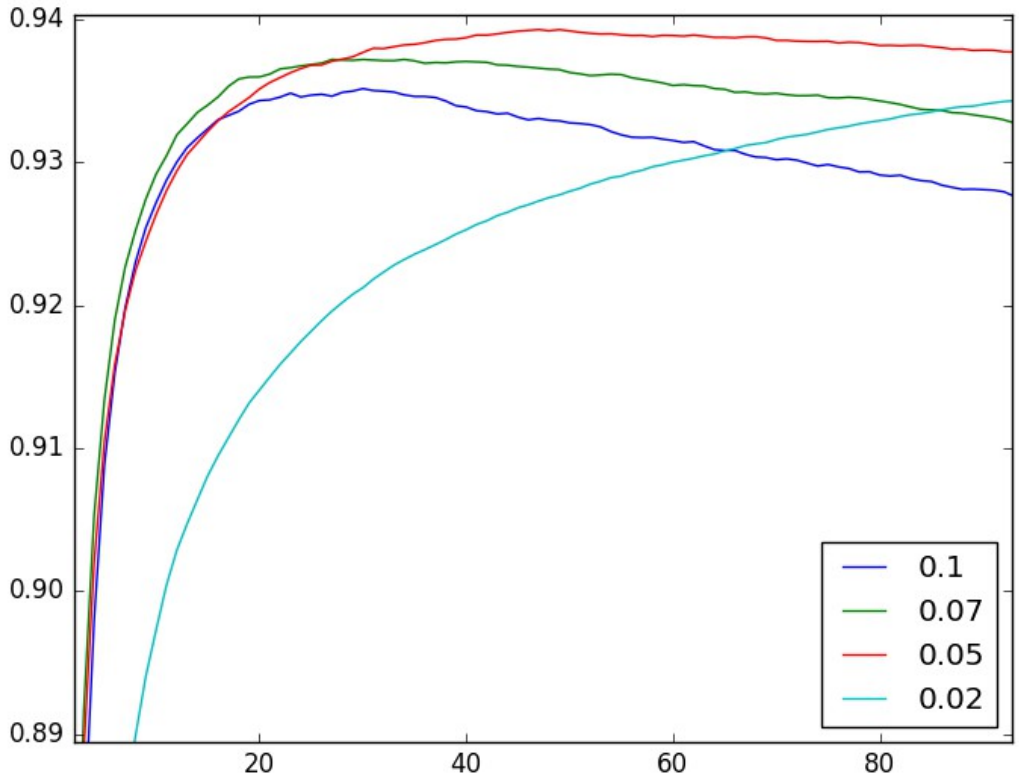

Figure 4.3: AUC for four different learning rate.

In Fig. 4.3, the X-axis represents the number of epoches and Y-axis represents the AUC value. We start with learning rate 0.1 and then reduce it to 0.07, 0.05 and at last we stop at 0.02 because the maximum value of AUC drop significantly. As we can see in this figure, the smaller the learning rate is, the more iterations are needed to reach the maxima. Especially, if we set 0.02 as the learning rate, the system can not reach the maxima within 80 iterations. We can see the value 0.05 has the highest AUC value after 30 epoches. So, we take 0.05 as our learning rate.

3. Number of components. This is the dimensionality of the latent feature matrix. We continue to use the preset hyperparameter values except we already choose Adagrad

as the learning rate schdule and 0.05 as the learning rate. Usually, more latent components will consume much more computational time. A small number of latent factors would result in a low AUC level while a large number of latent factors would make the system run for a long long time. So we simply choose our starting point at 40 and increase 10 per time to see the difference in performance.

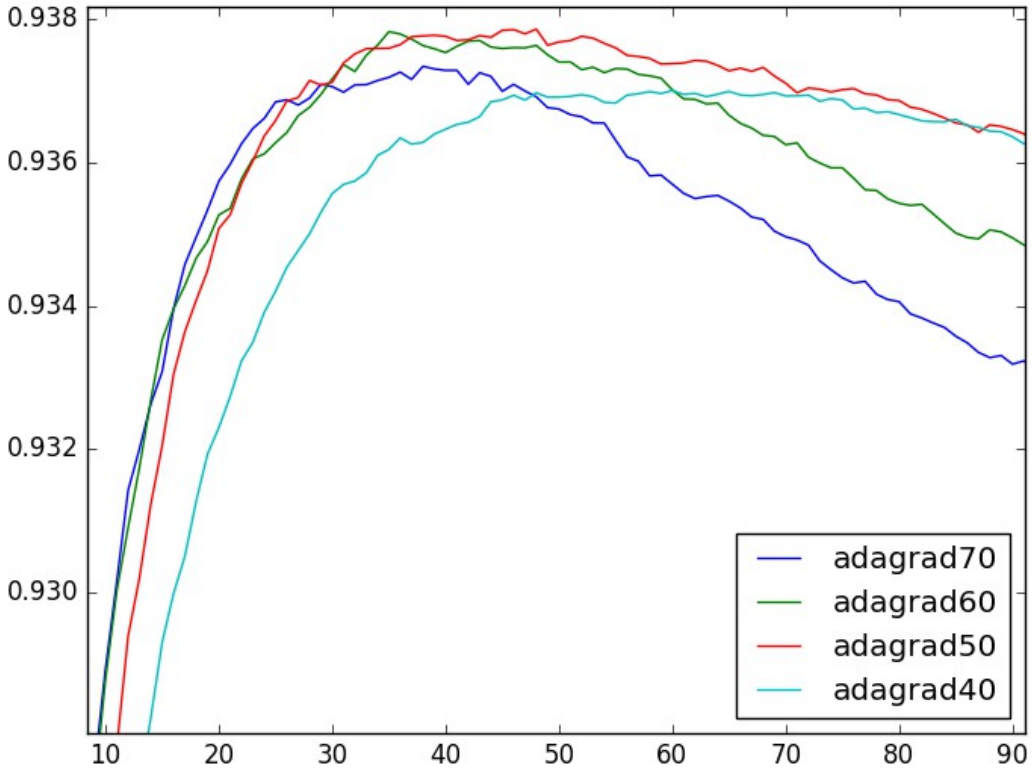

Figure 4.4: AUC for different number of components.

In Fig. 4.4, the X-axis represents the number of epoches and Y-axis represents the AUC value. We run the number 40 first and then 50, 60 and stop at 70 because the maximum value of AUC starts to drop when we use 70 as the number of components. The performances are close between numbers 50 and 60, but 50 components can maintain its performance in a high level for a longer time and drop milder with more iterations. So, we choose 50 as the number of components because it seems much

robustness than 60.

4. Before we run the experiment, we need to find out the best value for the $resta\_sim$. We run a demo test using WARP-KOS loss function in LightFM model and use the validation set to choose the value of w_1 and w_2. The precision that the system can achieve with different fractions of tr_sim is shown in the Fig.4.5. The X-axis represents the fraction percentage of tr_sim and Y-axis represents precision value. As we can see in this figure, when $x = 0$, it means the system only uses the hr_sim; while when $x = 100$, it means the system only uses the $tr - sim$. The performance of pure tr_sim is better than pure hr_sim is because the dataset is generated under the guidance of similarity based on restaurants' tags. However, as shown in the figure, the result that $resta\_sim$ beats the tr_sim represents the hr_sim can help the system to approximate the real situations.

   According to the result of precision, we should use 0.8 as w_2 and 0.2 as w_1.

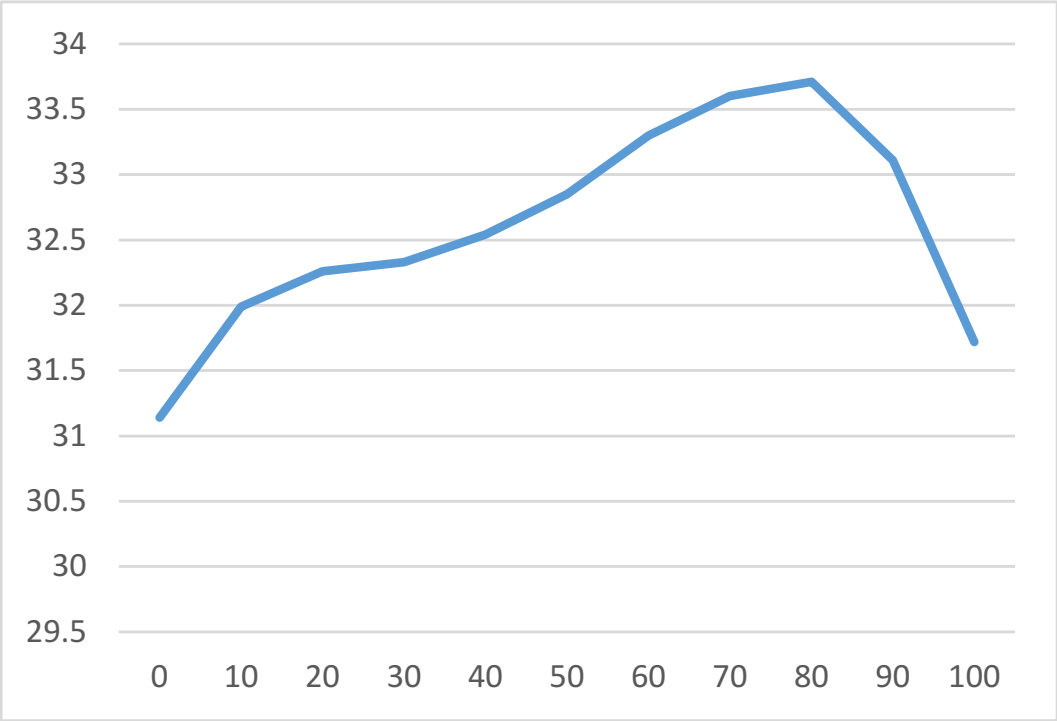

Figure 4.5: Precision on validation set with different fractions of tr_sim

5. For the k value of recall, we will choose five because the average amount of samples per user in a fold is lower than four. When we calculate the recall of those users that have fewer than five test samples, we will alternate the k value from five to each user's sample amount.

4.3 Results and Analysis

We use the above preset parameters to run three different pairwise learning methods in LightFM and compare to other five classic models.

Table 4.1: Precision comparison in LightFM model

	Similarity	Precision	PRRF
BPR	hr_sim	0.288	0.2066
	tr_sim	0.2966	0.2115
	resta_sim	0.3211	0.2354
WARP	hr_sim	0.3108	0.2259
	tr_sim	0.3256	0.2414
	resta_sim	0.3377	0.2579
WARP-KOS	hr_sim	0.3107	0.2272
	tr_sim	0.3306	0.2495
	resta_sim	0.3353	0.2624

We can find out that *resta_sim* outperforms the other two similarity approach by approximately 2% to 4%. In the meantime, *tr_sim* performs slightly better than the *hr_sim* as we expect because the *tr_sim* shares the same ideology with the calculation of similarity in Entrée system. Also, we can find out that even *resta_sim* can not improve the precision much from *tr_sim* in two WARP models, it still has 2% increase in PRRF. In another word, the *resta_sim* performs better than *tr_sim* to provide the accurate recommendation in the first shot.

Table 4.2: Recall and F-score comparison in LightFM model

	Similarity	Recall@5	F-score
BPR	hr_sim	0.4463	0.3500
	tr_sim	0.4656	0.3624
	resta_sim	0.4716	0.3820
WARP	hr_sim	0.4632	0.3720
	tr_sim	0.4765	0.3868
	resta_sim	0.4811	0.3968
WARP-KOS	hr_sim	0.4588	0.3745
	tr_sim	0.4643	0.3862
	resta_sim	0.4756	0.3933

As shown in Table 4.2, *resta_sim* outperforms the others various from 2% to approximately 4% in Recall@5 whilst various from 2% to 3% in F-score. Since F-score balances the performance of precision and recall@5, it is a weighted average of these two metrics.

Table 4.3: Ranking relevant metrics comparison in LightFM model

	Similarity	MAP	MRR	AUC
BPR	hr_sim	0.7042	0.7809	0.8924
	tr_sim	0.7353	0.7933	0.8990
	resta_sim	0.7647	0.8268	0.9056
WARP	hr_sim	0.7258	0.8333	0.9406
	tr_sim	0.7635	0.8680	0.9578
	resta_sim	0.7759	0.8720	0.9655
WARP-KOS	hr_sim	0.7133	0.8426	0.9443
	tr_sim	0.7737	0.8630	0.9565
	resta_sim	0.7814	0.8860	0.9623

In Table 4.3, all three metrics are focusing on evaluating the ability to tell the positive from negatives. *resta_sim* is still the best similarity approach in these evaluations. It outperforms the others by over 6% in MAP over 4% in MRR while AUC's improvement is within 2%.

All three different pairwise approaches yield good results. And all three learn to rank methods in LightFM can achieve the best results while using *resta_sim*. Two WARP models perform better than the BPR because WARP uses a more flexible mechanism for updating the weight or latent factors in the algorithms.

Usually, the WARP-KOS will beat WARP, but since we have only three positive samples per user in training set, different k value will not affect the result much. Although WARP-KOS method does not outperform WARP in precision recall and F-score, it does perform slightly better in AUC and MRR evaluation metrics. Moreover, these two metrics show

that the k-order has better ability to rank the positive item higher in the recommendation list.

The improvement of using *resta_sim* is significant and in order to show how well the learn to rank methods can handle implicit feedback, we choose WAPR-KOS on behalf of the LightFM models to compare with five other classic models. The results are reported in the Table 4.4.

Table 4.4: Results of different Recommendation algorithms

	Precision	Recall@5	F-score	AUC	MRR	MAP
UBCF	0.1404	0.2344	0.1756	0.6856	0.4456	0.3867
CBR (Entrée System)	0.2604	0.4506	0.326	08546	0.677	0.6022
SBCF	0.2465	0.403	0.3058	0.8464	0.6456	0.5832
OHBCF (Entrée C System)	0.2809	0.4234	0.3377	0.9046	0.7263	0.6843
SVD++	0.2783	0.2873	0.2827	0.8846	0.7066	0.6366
LightFM	0.3353	0.4756	0.3933	0.9623	0.886	0.7814

As we expect, the UBCF is beaten by all other models in every single evaluation method. It is because Entrée system is a knowledge-based system and there is no rating in the original dataset. The ratings that UBCF uses are generated by the process of transforming behaviors into scores. This process cannot precisely reflect users' personal preference. Interestingly, if we count the "Browse" action to be the positive one, UBCF will have a better result. But this is against the meaning of browsing in reality and hard to explain or expand the usage of this model.

CBR outperforms the all other models except for the LightFM and the OHBCF. It is reasonable because CBR and the original system are working in the same ideology. The recommendations are decided by the similarities between restaurants. There are no new factors, which means no new noises too. Models SBCF, OHBCF and even LightFM are

all adding new factors and in the mean time with new noises together. This is a trade off process.

The results of SBCF show that it has little bit worse performance than the CBR model. That makes sense because the sparse similarity uses user behaviors only and ignores what characteristics restaurants have.

The results of model OHBCF ranks second in precision among all six models. Benefiting from the heuristic similarity process, it can beat the CBR model by relatively 8%. In the mean time, because of the noises adding in along with this heuristic similarity, such as those "half connection" behavior pairs that mentioned in section 3.3.2, the performance of recall drops by relatively 6% compared to the CBR model. To sum up, this trade off is not so cost effective with that extra running time and more complex processes.

Model SVD++ obtained a good score at precision but a relatively bad score at recall. This is because our dataset is imbalanced and the model overfits the training set, where only a small fraction is positive examples.

The LightFM model performs the best in all six evaluation metrics. This model solves the overfitting problem in SVD++ model in some extent because the LightFM model was trained by pairs of restaurants and updated until wrong predictions occurred. The performances in MAP and MRR show that the positive restaurants can be top ranked, which means the combination of *resta_sim* and learning to rank approaches can provide accurate recommendations with only one or two interactions.

Comparing to the best performance of the other models, LightFM model increases relatively by 19.3% in precision, 18.0% in recall, 16.5% in F-score, 6.4% in AUC, 22.0% in MRR and 14.2% in MAP. These large amounts of improvement show pairwise approaches of LTR model are good at dealing with implicit feedback, and heuristic similarity process can maximize the information we can learn from the real world dataset.

4.4 Summary

In this chapter, we run several tests to see whether the *resta_sim* is a better approach and to what extent this approach can help the system provide better recommendations. In particular,

- We introduced seven evaluation metrics and one of them is a customized metric for datasets like Entrée.
- We used those evaluation metrics to analyze different models including three learning to rank models in LightFM and five classic models.
- The experimental results showed that *resta_sim* achieved the best performance in the comparison within the LightFM models.
- Also, the WARP-KOS approach to learning to rank had better performance in the comparison between the LightFM models and the other five classical models.

Chapter 5

Discussion

In the real-world recommendations, learning from users' implicit feedback is important because users can not always provide enough explicit feedback for different reasons, such as laziness or privacy concerns. The closer the gap between a user profile and his/her real characteristics, the more accurate the recommendation we can provide.

This thesis presents an approach to making use of users' behaviors and it is encouraging to know that the pairwise LTR models and user behavior mining achieve excellent performance on the Entrée dataset. This kind of transformation from user behaviors to ratings can be used in different domains. Also, the method of analyzing the relationship between user behaviors can be used in different domains. For example, in the music recommendation domain, developers can create a tagging system to allow users to share their feelings by tagging the music they heard, and then analyze the relationship between tags such as, "mild" /"wild", "upset"/ "happy", and "love"/ "skip". The listening time of music, which is a implicit feedback, can be used to determine how much the user loves this song. Those tags and the listening time can form the context of the music dataset to which the process described in this thesis can be applied.

On the other hand, $sparse\_sim$ in equation 3.2 can be used in web browsing record-based recommendation systems such as paper recommendation or news recommendation. Restaurants can be tagged with different aspects such as the quality of service, price, cost performance, and type of food. Similarly, research papers can be tagged with the

information on multiple dimensions such as author, title, main idea, and the fields of research while news are much more intuitively that can be tagged with different types of news such as sports and finance.

Opposite behaviors always exist in human life. Learning a user's restaurant preferences not only helps to choose a nicer or a cheaper restaurant for him (or her) but also helps in the recommendation of more or less expensive products based on the user's incoming level. If a user always prefers a more expensive restaurant, we can recommend a more expensive product to him/her, and we can recommend the less expensive product to those who prefer cheaper restaurants. Similarly, if a user always prefers more traditional restaurants, we may recommend classical goods to him/her; otherwise, we can recommend interesting fashion things like pop music or clothes with a style brand. We can make use of these opposite preferences as we do in calculating the heuristic similarity in section 3.3.2. We can use these approaches to making cross-domain recommendations.

Our heuristic similarity approach in section 3.3.2 can be used to reveal the real connections between users and restaurants. With the help of mapping the behaviors described in section 3.1 into score and the consideration of inner-connection between those behaviors and the rating preferences of users, we can improve the heuristic similarity approach, which would become the weighted similarity approach in section 3.4.2 to represent a much more precise similarity calculation between restaurants. Therefore, we can build a much more powerful recommender system. Using the features of restaurants to make recommendations can solve the cold-start problem in some extend. Because if we want to add a new restaurant, we simply put it in to the database and give this restaurant some tags to represent its characteristics. And more over, using the navigation behavior to build the user profile is supplementary to the traditional building methods. The more detailed the user profile we have, the more accurate the recommendation we get.

In this thesis, we compare three pairwise approaches as well as five classical models on Entrée dataset. Such a comparison shows that more flexible updating mechanisms, which are those two WARP approaches, could simulate the real situation better. Moreover, pairwise approaches focus on ranking rather than just ratings, which could perform better

and satisfy users easier by returning a list of recommendations rather than displaying predicted ratings of recommendations.

Chapter 6

Conclusions and Future Work

In this chapter, we provide a summary of our work in this thesis. Then, we will propose a number of future research directions.

6.1 Contributions

1. We analyze the dataset of the Entrée system carefully and quantify both implicit and explicit feedback (also known as user behaviors) into ratings. These ratings can help us to understand the meaning of the user behaviors much more intuitively.

2. We make use of the opposite behavior pairs and perform context-aware approach to computing a modified heuristic similarities between users and restaurants.

3. We manage to minimize the ten-dimensional context vector into four dimensions. With this reduction of the context representation, we can save some running time of each experiment and reduce memory consumption.

4. We take the tag set into account, which represents characteristics of restaurants, and propose a weighted similarity approach to balancing the effect on the weighted restaurants similarity between the tag similarity and the heuristic similarity.

5. We propose a modified evaluation metric to calculate the portion of positive samples rather than using the original top-k precision method, which helps distinguish the performance difference between algorithms.

6. We use precision (the fraction of positive), recall, F-score, mean reciprocal rank (MRR), mean average precision (MAP) and area under curve (AUC) to evaluate the performances between three different pairwise approaches of learn to rank method based on matrix factorization and five classic recommenders including UBCF, content-based recommender (CBR), sparse behavior collaborative filtering (SBCF), original heuristic behavior collaborative filtering (OHBCF), and singular value decomposition plus plus (SVD++).

7. The work in this thesis has been exposed to the research community of the field and resulted in one book chapter [13] and one journal submission [14].

6.2 Conclusions

The whole world is on the trend of the third industrial revolution leading by the Internet. It is changing the way of peoples thinking, their habits of life and the relationship between them, and is building unprecedented business and social models. Nowadays, the scale of information is growing in geometric exponential level, but our ability to obtain information remains the same as our ancestors. The information explosion has made our life comfortable but also created heavy work for us to find the things we want. The emergence of recommender systems meets our need in this new era.

Recommender systems, information retrieval, and user characteristic and item feature analysis are becoming a trend in Internet development. Of course, there are a lot of problems waiting for solutions such as:

- User information collection is over-dependent on the explicit feedback especially the ratings.

- Lack of automatic analysis on implicit feedback.

- A large portion of algorithms are based on collaborative filtering and lack of solutions for sparse datasets or missing value datasets.
- Lack of efficient ensemble models or integrated algorithms on applications.

To overcome these problems, this thesis carried out some theoretical researches, and try to use new approaches to deal with log type data. In the third chapter, we transformed both implicit and explicit context (user behaviors) into ratings which were then fed into the recommender systems as input, and we have performed context-aware approach to compute the heuristic similarities. We managed to reduce the dimension of context vector from ten to four. Moreover, we proposed a weighted restaurant similarity using both tag and heuristic similarities.

In the fourth chapter, we ran some experiments on the Entrée dataset and we used several evaluation methods to compare the performance between three different pairwise approaches to ranking based on matrix factorization and five classic recommenders. Finally, we figured out $resta\_sim$ performed better than hr_sim and tr_sim whilst pairwise approaches to learning to rank could outperform other classic models by a large margin.

This thesis supports that pairwise approaches are good at dealing with implicit feedback, and heuristic similarity process help the system to restore real-world information and provide good recommendations.

6.3 Future Work

This thesis focuses on analyzing the preference of user behaviors and leveraging the historical dataset in the recommender system. Although our data scrubbing process and the transformation of behaviors to ratings help the recommender system achieve good results. But there are still two shortcomings needed to be addressed in the future.

1. It is based on the assumption that the exit point is a positive rating. This assumption is the same as that we assume most users are satisfied with the recommendations that

they receive. But we can not rule out the possibility that some users are abandoning their searches in frustration. In this case, the dataset would have some noisy samples.

2. We assume the interactions made from the same IP address comes from the same user. This assumption would make the dataset noisy too because if IP addresses represent public devices, we may have different users with the same IP address.

Also, in building a restaurant reommender system, this system should have the ability to firstly find out all the candidates that match the user's navigate operation and then not only rank them according to the similarities but also consider the average ratings of this restaurants given by all other users in the database. It is intuitively that if a restaurant always receives negative ratings and has a bad reputation, we should not recommend this kind of restaurants to users.

Moreover, the navigation action sequence may be another possible further extension of the system. For example, consider two users, whose actions are the same on each rated restaurants. In our similarity calculations, we will consider they have the same preferences so that their similarity is one. But it means their preferences have some differences if they have different action sequence in the real world. Therefore, we can take the navigation action sequence into account.

Last but not least, the algorithm we use in this thesis can be only run on a single computer which makes it harder to deal with massive data if we enlarge the usage of this recommender. The recommender system can work on a server that can be modified to be either a single computer or a cluster of computers. Hence, we would like to improve the parallel computation ability of this algorithm.

References

[1] Pattie Maes et al. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[2] Starr R Hiltz and Murray Turoff. Structuring computer-mediated communication systems to avoid information overload. *Communications of the ACM*, 28(7):680–689, 1985.

[3] Kaggle. Kaggle. `http://Kaggle.com`, 2010. Accessed August 4, 2017.

[4] Netflix. Netflix prize. `http://netflixprize.com`, 2007. Accessed August 4, 2017.

[5] Yahoo. Kdd. `http://www.kdd.org/kdd-cup#`, 2007. Accessed April 4, 2017.

[6] Amazon. Amazon. `http://amazon.com`, 1994. Accessed August 4, 2017.

[7] Ebay. Ebay. `http://ebay.com`, 1995. Accessed August 4, 2017.

[8] Gary S Figiel, Charles Epstein, William M McDonald, Jody Amazon-Leece, Linda Figiel, Aida Saldivia, and Susan Glover. The use of rapid-rate transcranial magnetic stimulation (rtms) in refractory depressed patients. *The Journal of neuropsychiatry and clinical neurosciences*, 10(1):20–25, 1998.

[9] Mark S Fox and Stephen F Smith. Isisa knowledge-based system for factory scheduling. *Expert systems*, 1(1):25–49, 1984.

[10] Suresh Kumar Gorakala. *Building Recommendation Engines*. Packt Publishing, 2016.

[11] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[12] Robin D Burke, Kristian J Hammond, and BC Yound. The findme approach to assisted browsing. *IEEE Expert*, 12(4):32–40, 1997.

[13] Haoxian Feng and Thomas Tran. Context-aware approach for restaurant recommender systems. In *Encyclopedia of Information Science and Technology, Fourth Edition*, pages 1757–1771. IGI Global, 2018.

[14] Haoxian Feng and Thomas Tran. User behavior learning in designing restaurant recommender systems: An approach to leveraging historical data and implicit feedback. In submission to Knowledge and Information Systems (KAIS), August 2017.

[15] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[16] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[17] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.

[18] Digg. Digg website. `http://digg.com`, 2008. Accessed April 4, 2017.

[19] Digg. Digg recommendation engine updates. `http://about.digg.com/blog/digg-recommendation-engine-updates`. Accessed April 4, 2017.

[20] Paul Horton and Kenta Nakai. Better prediction of protein cellular localization sites with the it k nearest neighbors classifier. In *Ismb*, volume 5, pages 147–152, 1997.

[21] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[22] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[23] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David 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. ACM, 2002.

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

[25] Michael Pazzani and Daniel Billsus. Content-based recommendation systems. *The adaptive web*, pages 325–341, 2007.

[26] Bernardo Magnini and Carlo Strapparava. Improving user modelling with content-based techniques. In *International Conference on User Modeling*, pages 74–83. Springer, 2001.

[27] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[28] Peter W Foltz. Using latent semantic indexing for information filtering. In *ACM SIGOIS Bulletin*, volume 11, pages 40–47. ACM, 1990.

[29] Sofiane Abbar, Mokrane Bouzeghoub, and Stéphane Lopez. Context-aware recommender systems: A service-oriented approach. In *VLDB PersDB workshop*, pages 1–6, 2009.

[30] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[31] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[32] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[33] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[34] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[35] Hsin-Hsien Lee and Wei-Guang Teng. Incorporating multi-criteria ratings in recommendation systems. In *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, pages 273–278. IEEE, 2007.

[36] Fuzheng Zhang, Nicholas Jing Yuan, Kai Zheng, Defu Lian, Xing Xie, and Yong Rui. Exploiting dining preference for restaurant recommendation. In *Proceedings of the 25th International Conference on World Wide Web*, pages 725–735. International World Wide Web Conferences Steering Committee, 2016.

[37] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[38] Daniel Herzog and Wolfgang Wörndl. A travel recommender system for combining multiple travel regions to a composite trip. In *CBRecSys@ RecSys*, pages 42–48, 2014.

[39] Rahim A Abbaspour and Farhad Samadzadegan. Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications*, 38(10):12439–12452, 2011.

[40] Yohei Kurata, Yasutaka Shinagawa, and Tatsunori Hara. Ct-planner5: a computer-aided tour planning service which profits both tourists and destinations. In *Workshop on Tourism Recommender Systems, RecSys*, volume 15, 2015.

[41] Guan-Shen Fang, Sayaka Kamei, and Satoshi Fujita. A japanese tourism recommender system with automatic generation of seasonal feature vectors. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 8(6):347–354, 2017.

[42] Kenta Oku, Shinsuke Nakajima, Jun Miyazaki, and Shunsuke Uemura. Context-aware svm for context-dependent information recommendation. In *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*, pages 109–109. IEEE, 2006.

[43] Cataldo Musto, Giovanni Semeraro, Pasquale Lops, and Marco de Gemmis. Contextual evsm: A content-based context-aware recommendation framework based on distributional semantics. In *International Conference on Electronic Commerce and Web Technologies*, pages 125–136. Springer, 2013.

[44] Cataldo Musto, Giovanni Semeraro, Pasquale Lops, and Marco Gemmis. Random indexing and negative user preferences for enhancing content-based recommender systems. *E-Commerce and Web Technologies*, pages 270–281, 2011.

[45] Magnus Sahlgren. *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, 2006.

[46] Yoon Ho Cho, Jae Kyeong Kim, and Soung Hie Kim. A personalized recommender system based on web usage mining and decision tree induction. *Expert systems with Applications*, 23(3):329–342, 2002.

[47] Karen HL Tso-Sutter, Leandro Balby Marinho, and Lars Schmidt-Thieme. Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1995–1999. ACM, 2008.

[48] Chao Wu and Bo Zhou. Analysis of tag within online social networks. In *Proceedings of the ACM 2009 international conference on Supporting group work*, pages 21–30. ACM, 2009.

[49] Sharon Givon and Victor Lavrenko. Predicting social-tags for cold start book recommendations. In *Proceedings of the third ACM conference on Recommender systems*, pages 333–336. ACM, 2009.

[50] Denis Parra and Peter Brusilovsky. Collaborative filtering for social tagging systems: an experiment with citeulike. In *Proceedings of the third ACM conference on Recommender systems*, pages 237–240. ACM, 2009.

[51] Shilad Sen, Jesse Vig, and John Riedl. Tagommenders: connecting users to items through tags. In *Proceedings of the 18th international conference on World wide web*, pages 671–680. ACM, 2009.

[52] Frederico Durao and Peter Dolog. A personalized tag-based recommendation in social web systems. *arXiv preprint arXiv:1203.0332*, 2012.

[53] Sogol Naseri, Arash Bahrehmand, and Chen Ding. An improved collaborative recommendation system by integration of social tagging data. In *Recommendation and Search in Social Networks*, pages 119–138. Springer, 2015.

[54] Sarabjot Singh Anand and Bamshad Mobasher. Contextual recommendation, from web to social web: Discovering and deploying user and content profiles: Workshop on web mining, webmine 2006, berlin, germany, september 18, 2006. revised selected and invited papers, 2007.

[55] Konstantin Bauman and Alexander Tuzhilin. Discovering contextual information from user reviews for recommendation purposes. In *CBRecSys@ RecSys*, pages 2–9, 2014.

[56] Daniel Billsus and Michael J Pazzani. Learning collaborative information filters. In *Icml*, volume 98, pages 46–54, 1998.

[57] S. Funk. Netflix update: Try this at home. `http://sifter.org/~simon/journal/20061211.html`, 2006. Accessed April 4, 2017.

[58] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 931–940. ACM, 2008.

[59] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[60] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[61] Xin Guan, Chang-Tsun Li, and Yu Guan. Enhanced svd (esvd) for collaborative filtering.

[62] Y Koren. Factorization meets the neighborhood: a multifaceted factor in the neighbors: Scalable and accurate collaborative filtering. In *Proc. 14th ACM Int. Conference on Knowledge Discovery and Data Mining (KDD 2008). ACM Press*, 2008.

[63] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[64] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199. ACM, 2008.

[65] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

[66] Gábor Takács and Domonkos Tikk. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 83–90. ACM, 2012.

[67] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90. ACM, 2010.

[68] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[69] Joseph Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, pages 783–791, 1985.

[70] Arthur Fine. Hidden variables, joint probability, and the bell inequalities. *Physical Review Letters*, 48(5):291, 1982.

[71] David J Hand and Robert J Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine learning*, 45(2):171–186, 2001.

[72] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.

[73] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.

[74] Jason Weston, Hector Yee, and Ron J Weiss. Learning to rank recommendations with the k-order statistic loss. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 245–248. ACM, 2013.

[75] Maciej Kula. Metadata embeddings for user and item cold-start recommendations. In Toine Bogers and Marijn Koolen, editors, *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.*, volume 1448 of *CEUR Workshop Proceedings*, pages 14–21. CEUR-WS.org, 2015.

[76] Susanne Carroll and Merrill Swain. Explicit and implicit negative feedback. *Studies in second language acquisition*, 15(03):357–386, 1993.

[77] Zhiwen Yu, Yuichi Nakamura, Seiie Jang, Shoji Kajita, and Kenji Mase. Ontology-based semantic recommendation for context-aware e-learning. *Ubiquitous Intelligence and Computing*, pages 898–907, 2007.

[78] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 154–161. Acm, 2005.

[79] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[80] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.