VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Food Recommendations

Hai Quang Tran : 300224467

Supervisors: John Clegg, Dr Xiaoying Gao, Dr Ian Welch

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

The goal of this report is to demonstrate what has been achieved in the first three months of the Food Recommendations project and outline the working plan for the remainder of the project.

The project looks at how a recommender system can be used to recommend food dishes to users based on their personal preferences, their previous actions such as likes/dislikes, and other factors such as their location and price. In particular, this project explores collaborative filtering (CF) techniques by leveraging existing research and explores these algorithms to best suit the use case of recommending food to users.

The literature review investigates existing recommender systems that use CF techniques and compares the advantages and disadvantages of each technique to identify suitable CF techniques for our use case.

Suitability of these algorithms will be considered on factors that are important for a web-centric mobile application called 'What's On The Menu' currently being developed. The recommender system will have to account for a range of factors from the user experience, the accuracy of the recommendation system, and the speed/performance that the system can provide recommendations to users.

Contents

Chapter 1

Introduction

With the vast food options around us, it is often difficult to decide what to eat. On some days we may feel like eating our favourite foods, but on other days we could feel more explorative, wanting to try something new. There are many factors that determine our decisions on what we want to eat such as our preferences, our current mood, and what food providers are nearby. Other factors that may contribute is the price of the food dish, the cuisine type, the meat type, and so forth. Additionally, some of us may have food intolerances which restrict our food options.

Whats On The Menu (WOTM) is a web-centric mobile application that is focused on recommending food dishes to users based on their personal preferences, previous actions such as likes/dislikes, and other factors such as location, price, and so forth. These personalised recommendations are important as it saves time for users and allows users to explore preferred food options they otherwise might not have discovered alone.

Existing applications such as Yelp and Foursquare have similar concepts but lack personalised recommendations for specific food dishes. These applications focus on a broader scope such as popular restaurants, coffee places, activities, and so forth. This can often lead to a cluttered interface, affecting the user experience, and may provoke confusion in users. Food spotting is an application that focuses on food dishes but does not provide personalised recommendations to users. Instead, they show popular food dishes and focus on crowd-sourcing, relying on users to upload food dishes. What's On The Menu (WOTM) differs as the goal is to provide personalised food recommendations to users through collaborative filtering.

Collaborative filtering is a popular recommendation technique used to recommend items to users based on previous rating patterns and the behaviours of others [1, 2, 3].

The goal of this project is to find which types of collaborative filtering algorithms best suit the WOTM application. Suitability of these algorithms will be considered on factors that are important for WOTM. The recommender system will have to account for a range of factors from the user experience, the accuracy of the recommendation system, and the speed/performance that the system can provide recommendations to users. This project aims to explore this, by implementing different types of collaborative filtering techniques to recommend food to users based on their personal preferences and previous behaviours.

1.1 What's On The Menu Overview

What's On The Menu (WOTM) is an existing system that aims at providing personalised food recommendations to users. It is currently in development and was created by John Clegg. WOTM consists of two components: WOTM Web and WOTM API. WOTM Web is the front-end of the application that deals with interactions in the client browser. WOTM API is the back-end of the application that deals with the application logic and management of data. We will be extending the existing system by creating an additional component called WOTM Recommendations. This component will connect to the other components to provide recommendations to the users, and will manage anything related to the recommendation system.

Chapter 2

Background Survey & Related Work

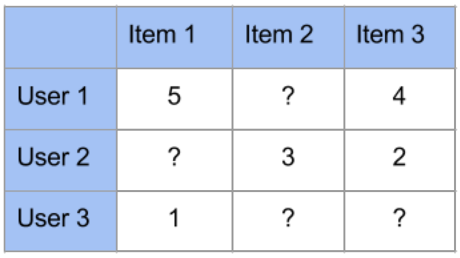
2.1 Collaborative Filtering

Collaborative filtering (CF) is a popular technique that has been successfully used in recommender systems [1, 2, 3]. The main idea behind collaborative filtering is that rating behaviours of others can be used to predict items that the active user will be interested in. Intuitively, this algorithm stems from the assumption that users having similar opinions for items that they have both rated tend to have similar tastes to each other, having similar opinions about other items [2]. For example, most of us have friends with similar opinions and tastes to us. Perhaps you and a friend both like eating chicken sandwiches. Since you and your friend have similar tastes, your friend may then be able to suggest to you another food dish that you might enjoy such as duck with rice. Collaborative filtering uses the same concept, but usually on a larger scale.

Users express their preferences for a variety of items through ”ratings” which can be in different forms such as 1-5 star ratings, or binary scale ratings such as likes/dislikes. These ratings are typically represented as a (User, Item, Rating) triplet. These ratings can be represented in the form of a matrix, where the rows of the matrix are users, the columns of the matrix are items, and the ratings are represented inside the matrix (See Figure 2.1). Since the matrix contains sets of rating triplets, the (User, Item) pairs will exist where a user has not yet rated the item, thus the matrix is usually sparse.

Since the matrix can be sparse, collaborative filtering can struggle to address new items and users because there are not enough ratings to provide accurate recommendations to these users. Sparse ratings can also lead to another issue called the ’Cold Start’ problem, where ratings for these new items or new users have not yet been collected.

Using the matrix, a list of top N recommendations is usually given to the user representing the items they will like, computed through collaborative filtering techniques. For



	Item 1	Item 2	Item 3
User 1	5	?	4
User 2	?	3	2
User 3	1	?	?

Figure 2.1: A user-item ratings matrix containing scaled ratings from 1-5.

example, the active user may be recommended 10 food dishes that might be of interest to them whenever they log into WOTM.

There are three main areas of CF, these areas are memory-based CF techniques, model-based CF techniques, and hybrid-based CF techniques. In this project, we will be examining in particular, the neighbourhood methods which are a form of memory-based CF techniques, and latent factor models which are a form of model-based CF techniques [3, 4]. Neighbourhood methods, and Latent factor models will be explained in the following sections.

2.2 Neighbourhood Methods

Neighbourhood CF methods can be classified into the class of memory-based CF techniques because they use the entire or a sample of the ratings matrix to find the similarity between items or users through a heuristic [5, 2]. The main idea of this approach is to use the ratings matrix to compute the similarity between users or items which are then used for recommendations. This typically involves finding all user-user similarities or item-item similarities and using neighbourhood algorithms such as K-Nearest Neighbour to find the K most similar items or users, referred to as neighbours. The K most similar items or K most similar users are then used to predict how the active user will feel about specific items, thus being able to make recommendations for the active user based on their neighbours. Since these neighbours may contain quirky characteristics that are not common among the active user, methods tend to take the weighted average of the neighbours ratings or simple weighted average to generate predictions for the active user [3].

Common similarity measures that are used are Pearson's Correlation, Cosine similarity, Euclidean distance, Jaccardian similarity and so forth, finding similar users to the active user or similar items that the active user has previously performed actions on. The most common neighbourhood method that is used to find the most similar users or items is the K-Nearest Neighbour algorithm. This algorithm finds the K-nearest neighbours based on a heuristic, in this case, the similarity measure. Other neighbourhood algorithms are K-Means, K-d Trees, and Locality Sensitive Hashing [3].

The following section explains the difference between user-based CF and item-based CF.

2.2.1 User-based Collaborative Filtering

User-based CF uses a similarity measure to find like-minded individuals that are similar to each other, producing recommendations to the active user based on ratings of similar users. The defining characteristic of user-based CF is based upon similarities between users [6]. A concrete example would be if two users rated the same food dishes similarly. The first user has indicated that they like the Big Mac and the Whopper Burger. Similarly, the second user indicated that they like the Big Mac too. Therefore, since these two users previous ratings are similar (both like Big Macs), they are to be considered similar. We can then recommend to the second user that they should try the Whopper Burger, since the users have similar opinions about the items. User-based CF has the same premise but typically on a larger scale - more ratings, more users, and more items are involved.

2.2.2 Item-based Collaborative Filtering

Another approach that is commonly used is item-based CF. Instead of finding users that are similar to each other, item-based CF focuses on previous behaviour of the active user and recommends items that are similar to these items [6]. Similar items are extracted by the rating patterns of other users rather than the attributes of the item - Two items are considered similar if users rate them similarly [2]. For example, a user could have previously liked the following dishes: a chicken sandwich, chicken nuggets, and chicken soup. Item-based CF will find similar items that the user has previously rated by using a similarity measure on their preference history to find other items. By using this technique, it may recommend items based on the users previous interests, suggesting new items such as chicken salad, or a chicken burger since users who like the previous dishes of the active user tend to like these dishes. This is the main idea behind item based collaborative filtering.

2.3 Latent Factor Methods

Latent factor models can be classified into the class of model-based CF techniques because they try and explain ratings by automatically inferring meaningful information from users previous rating patterns which are then used to make intelligent predictions based on this meaningful information [3]. For example, these methods automatically try and learn about the properties or characteristics of the items such as the food dishes, and learn how much the user likes each of these properties, otherwise known as latent factors [7].

Matrix factorization is seen as one of the successful techniques of latent factor models [5, 4]. Singular Value Decomposition is a matrix factorization technique that is well-known and can be applied to the CF domain, identifying latent semantic factors [4]. The following section talks about the matrix factorization technique of Singular Value Decomposition and how it is used to recommend items to users.

2.3.1 Singular Value Decomposition

The main goal of Singular Value Decomposition (SVD) is to train a model to learn what properties of a dish that a user likes. This can be used to predict how a user may feel about another dish by looking at the properties that are inferred from that dish.

SVD decomposes the ratings matrix, where both users and items are mapped to a joint latent factor space of dimensionality f - the space containing the inferred properties of the food dishes, and how the users feel about those properties. The latent space tries to explain the ratings given from the ratings matrix by considering these inferred latent factors. These latent factors are represented in the form of an item vector $q_i \in \mathbb{R}^f$ and a user vector $p_u \in \mathbb{R}^f$ where i is a given item and u is a given user. The vectors contain a specified number of factors for each item and user, which are used to predict how the user may feel about other items that they have not yet rated [4]. Factors in the item vectors could be dimensions regarding the properties of the items. An example for food dishes are item vectors that discover factors measuring obvious dimensions in the food dish such as the ingredients, spice levels, cuisine type, or meat type; less defined dimensions such as carbohydrates, or uninterpreted dimensions in the food dish [4]. User vectors contain factors that measure the extent that the user likes those factors which are the properties that are inferred from the food dishes [4]. These factors in q_i and p_u can be positive or negative, representing how much the properties contained within each dish, and how much the user likes those properties.

These feature vectors contain a specified number of factors for each item and user, which are used to predict how the user may feel about other items that they have not yet rated [4]. For this model, a user's predicted rating for a dish would be the dot product of the food dish vector, and the user vector $q_i^T p_u$. This approximates to the active user's rating of the current item, denoted by r_{ui}, leading to the estimate [4]

$$\widehat{r} = q_i^T p_u \tag{1}$$

.

Since the user vector represents how much they like about the characteristics of the dish, and the item vector represents the portion of how much of the dish contains those properties [4], we can use the dot product - multiplying the dish containing a specific property from the item vector with how much the user likes that property from the user vector, and repeat this for all elements in the vectors to see the overall interest that the user has of the dish. This represents the active users interest about the overall dish and is used to estimate the rating a user will give to any item.

Now that we understand how the predicted ratings for an item and user are computed, you may be wondering how we infer the item and user vectors explained above.

To do this, the system learns the model by fitting the previously observed ratings. However, directly modelling the observed ratings can lead to over fitting, being overspecialized leading to inaccurate future predictions.

To prevent this, the idea is to use a regularized model that is used to generalize the previous ratings in a way that is able to predict future, unknown ratings. To learn the vectors for the items and users, the system uses an equation that minimizes the regularized squared error on the set of known ratings.

$$min_{q*,p*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u) + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2}$$

K represents the training set from the rating matrix where r_{ui} is known for the user, item pairs (u, i). The constant λ controls the regularization used to penalize the magnitudes to avoid over fitting, and is typically tuned by cross-validation. Minimization is performed by Alternating Least Squares [7].

Alternating Least Squares

The main purpose of Alternating Least Squares (ALS) is to find the minimum error from equation 2, used to learn the factors in each of the vectors based on the rating matrix. ALS works to find the item and user vectors by initializing the vectors with random values depending on how many factors are specified. From this, it fixes values of one of the vectors first such as the item vector, then computes and reassigns values of the user vector according to the minimization equation, shown as equation 2 above. After this, it alternates - the user vector values are now fixed, and the re-computation of the item feature values occur, trying achieve the result of the smallest value for the minimization equation. This technique keeps alternating between fixing the vector values, until there is a convergence where the values in the two vectors no longer change the the minimization equation, or there is small changes, hence the name Alternating Least Squares. This is used to find the minimum error from the equation mentioned above, to learn the factors in each of the vectors.

2.4 Literature Review

The term 'collaborative filtering' was first introduced in 1992 by Goldberg et al. to describe the technique used in Tapestry, one of the earliest known recommender systems [4, 8, 1, 3].

Tapestry [8] was created to handle electronic documents and used manual collaborative filtering, allowing users to query information based on others opinions about the documents. These opinions were in the form of annotations or replies which users were encouraged to make on documents to increase probability of relevant results returned from queries [2]. Tapestry relied on opinions from a small community such as an office work-group, where each person's opinion was trusted. Larger communities could not rely on every person knowing each other, leading to new collaborative filtering techniques being developed [1].

More recommender systems emerged as value was seen in the potential to increase sales from recommendations - customers may purchase suggested items that they might not have seen otherwise [2]. Perhaps the most popular recommender system in the late 1990's was used in Amazon.com, collecting user purchase history, browsing history, and recently viewed items to recommend items that the user may buy [2]. Other recommender systems consisted of Jester [9] for jokes, and Ringo [10] for music. GroupLens [11] were the first to introduce a neighbourhood collaborative filtering technique. Building upon the Tapestry concept, GroupLens created an automated user based collaborative filtering technique for recommending Usenet articles that users may be interested in. The advantage of neighbourhood methods is that they are intuitive, easy to implement, and produce highly effective results [3, 12]. Despite providing accurate recommendations, user-based collaborative filtering techniques were computed in real time and performance would degrade as more users and items were added to the system, leading to scalability and performance issues [13, 1, 14].

This required collaborative filtering techniques that could easily scale and still produce high quality recommendations leading to the exploration of item-based collaborative filtering. Item-based collaborative filtering techniques were developed to address scalability limitations of the user-based techniques [3]. Sarwar et al. analyzed various item-based recommendation algorithms, computing item-item similarities and comparing the accuracy with traditional KNN user based collaborative filtering techniques [1]. Sarwar et al. found that items remained fairly static in the system, whereas user behaviours and preferences would often change. Because items were found mostly static, it meant precomputation could occur for item similarities. By having precomputed item similarities, traditional item-based collaborative filtering can then be applied, thus performance and scalability would be increased [12].

Other techniques such as model based collaborative filtering have been investigated to overcome the performance and scalability issues. Well known model based techniques include Bayesian belief nets [15], clustering models [16], and latent semantic models [17]. These models are based on learning patterns from users previous actions to predict new items, and are expensive but can be built offline allowing high scalability. The resulting model is "very small, very fast, and essentially as accurate as nearest neighbor methods" [1]. Sarwar et al. found Bayesian networks to be practical in the context where user "preferences change slowly with respect to the model" [1]. However, these models are not suitable for environments where the user preference model should be updated rapidly or frequently. Since model based approaches do not have to compute similarity measures to form neighbourhoods, they tend to produce faster recommendations and outperform neighbourhood models in terms of accuracy of recommendations [18, 1].

Although collaborative filtering is considered to be one of the most successful approaches to recommender systems [3, 18], they suffer from the problem of data sparsity [18, 3, 1, 4, 7, 13]. Data sparsity is when only a small subset of user ratings on items are recorded, leading to a insufficient number of ratings to produce accurate recommendations. Data sparsity specifically tends to appear in the 'cold start' problem, where new items or new users are entered into the system, but not enough information is supplied to produce accurate recommendations since recommendations are based on common items or users [3].

To alleviate this problem, hybrid approaches were investigated that combined collaborative filtering and other recommender techniques such as content based filtering. Adomavicius and Tuzhilin suggested creating user profiles such that demographic information could be included in similarity measures to provide extra content to find similar users or items. This effectively makes use of content-based filtering where recommendations are produced based on the content and attributes of the items, learning what attributes the user likes [18]. Well-known hybrid techniques include content-boosted collaborative filtering [19], and personality diagnosis [20, 3]. Hybrid approaches were implemented to address the limitations of collaborative filtering and content-based filtering techniques [18], but have increased complexity leading to more expensive computations. Additionally, external information is needed about the content of the items which may not be available, thus making hybrid approaches impractical in certain scenarios [3]. Sarwar et al. found a different approach that used dimension reduction techniques such as Singular Value Decomposition, making sparse rating models more dense by reducing the dimensionality of the product space, thereby condensing the modelled ratings of users and producing less missing information [13].

In 2006, the Netflix Prize competition attracted interest in the field of recommender systems [3]. Netflix offered a $ 1 million dollar prize to the first team to improve their movie recommender system by 10%. This attracted interest in the research field of recommender systems. The team "BellKor in Pragmatic Chaos" won the competition in 2009 basing their solution on a combination of latent factor models and neighbourhood models [21, 3]. These models took into account many biases which improved the predication accuracy. Koren et al. were part of the winning team, and wrote a paper explaining how temporal effects, and user biases could be accounted for in latent factor models such as Singular Value Decomposition, making it superior to neighbourhood methods because of its flexibility [4]. Koren and Bell later published a paper about their findings and solutions to the Netflix Prize competition in [7] and [22].

2.5 Discussion of Literature Review

Existing research focuses on the scalability and prediction-accuracy of these collaborative filtering techniques. Literature suggests that latent factor models such as SVD produce better prediction-accuracy and solve the scalability issue. Since SVD is a model-based CF technique, it means expensive computation can be done offline, training the model. When the model is already trained, the performance of recommendations will be fast since precomputation has already occurred offline. This makes a model-based SVD approach the top candidate for WOTM.

However, model-based approaches need to be updated frequently. In terms of scalability, WOTM is not expected to contain anywhere near the number of users or items as existing recommender systems used by Netflix, Facebook, and Google. For this reason, neighbourhood methods may be a good choice, as they provide for the ability to explain the reasoning behind the recommendations, as well as give good prediction-accuracy results. Scalability should not be a top concern with this application. Therefore, real time computation may be a factor to consider. Therefore, this method also needs to be explored to see if it is suitable for WOTM.

It is evident that there has been an abundance of existing research on collaborative filtering techniques and ways to improve the prediction accuracy. However, Ekstrand et al. states these factors alone, do not contribute to making a good recommender system [2]. Instead, Ekstrand et al. states that recommendation is not a "one-size-fits-all problem" [2]. Specific tasks, information needs, and item domains represent unique problems for recommenders, and design and evaluation of recommenders needs to be done based on the user tasks to be supported" [2]. Similarly, Martin argues that the recommender algorithms is only one factor from many for providing recommendations to users. Martin explains that the user experience, data collection, and other problems which make up the whole of the recommender experience need to be considered [2, 23]. Cosley et al. concluded in [24] that much of the accuracy problem has been solved in recommender systems, however delivering these accurate predictions to users in a way that creates the "best experience for them remains an open problem" [24].

For this reason, this project focuses on the goal of providing a recommender system that fits the needs for the "What's On The Menu" application. This involves considering how users ratings will be collected, the user experience, the recommendation process, and what factors are considered to be important in the recommendation process such as the trade-offs between scalability, prediction accuracy, and performance.

A focus on the recommender system will be on how to make the user experience as easy as possible for users. The user experience affects the collection of ratings from users and the frequency in which users will rate food dishes, thus affecting the quality of recommendations. Another focus will be on the performance of providing recommendations to the users. In this case, prediction accuracy may be less important than the speed/performance of recommendations being produced. If the recommendation process is slow, users are less likely to continue using the application. On the other hand, if prediction accuracy is not accurate, then users will be recommended items that may not be of interest. A fair trade-off must be considered.

Chapter 3

Work Done

3.1 Limitations

The main limitations of this project is that it focuses on a mobile application, rather than a website application. This forces some limitations in terms of how users will interact with the system since with a mobile interface, the small form factor affects many changes which will be described later on. Since existing components of WOTM are done in the programming language Ruby, it means we are confined to this language. In addition, the data in the application has already been modelled, including the dishes, restaurants, users and so on. The only data that had not been modelled was the data required for the recommender system such as like/dislikes and preferences.

3.2 Design Decisions

3.2.1 Open source projects

This section identifies the open source projects that were considered for this project.

Open source projects can be utilised to fit the projects specific needs, as opposed to the alternative of starting from scratch. This saves time, allowing the focus to be on the CF techniques themselves. The open source community has a range of available projects. In particular, a range of machine learning libraries are available that incorporate various types of CF techniques. Spotify have an open-source project called Annoy [25] that uses a CF neighbourhood approach. Recommendable [26] also uses a CF neighbourhood approach and is written in Ruby, making it easy to setup with the existing WOTM components [26]. There were many other open-source projects such as EasyRec [27], Apache Mahout [6], Lenskit [28], and so forth, containing a range of CF techniques.

Eventually, this led to the discovery of PredictionIO, the main open-source project used in this project. The following section explains the reasons why PredictionIO was chosen.

PredictionIO

PredictionIO is an open-source machine learning server used to "build and deploy predictive applications in a fraction of the time" [29, 30].

Figure 3.1 represents how PredictionIO is made to integrate with existing applications [29]. Data such as rating events from WOTM is sent to the Event Server where it is stored on PredictionIO. PredictionIO uses a distributed database that is easily scalable. Data from the event server is then used by algorithms from the engines to learn recommendations. These

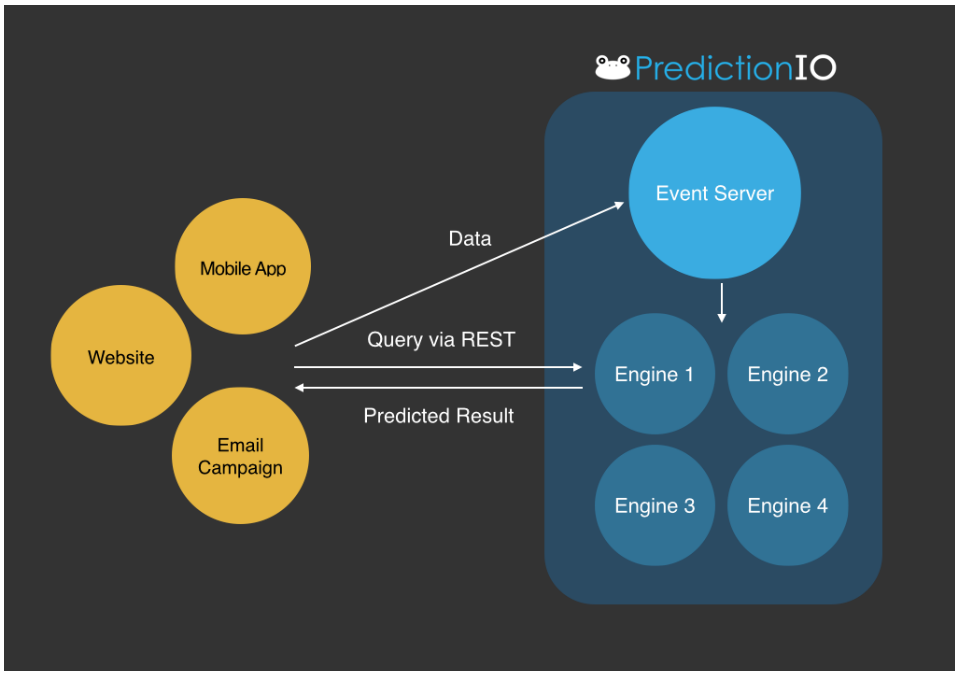


Figure 3.1: How existing applications interact with PredictionIO [29]

engines are deployed as distributed web services. WOTM can query these deployed engines to retrieve the top N recommendations for a specific user.

The main advantage of using PredictionIO is that engines can be easily swapped out making it easy to examine and evaluate various CF techniques. This saves time, switching the main focus on CF algorithms rather than configuring settings to accommodate various technology and tools.

Additionally, these engines implement a DASE architecture. DASE stands for Data, Algorithms, Serving, and Evaluator. These components are considered to be independent in the engine which allows for separation of concerns. Implementation of various algorithms can be applied at the Algorithms stage, and enables results from the various algorithms to be combined at the Serving stage to produce hybrid CF approaches for recommendations.

PredictionIO already contains a range of customizable engines, including engines that use SVD techniques for CF. The PredictionIO community is active and new engines are frequently being produced. There is concise documentation in addition to support available. For these reasons, we found PredictionIO to be our best candidate and decided to use it.

3.2.2 User Experience

With a mobile application it encourages user interaction that differs from that of a website. For instance, users are encouraged to use touch events to navigate through pages or to perform certain actions. This leads design decisions that accommodate such interactions, making it easier for users to perform tasks on mobile applications. The user experience of WOTM affects how rating data is collected from users, which can affect the recommendation system. The following section explains the design decisions for collecting events from users, and how these events can affect the recommender system.

3.2.3 Explicit Feedback

Recommender systems rely on different types of input data [4]. Explicit data is referred to as a direct record of someones interest of an item. For example, Netflix collects star ratings for movies, and TiVo users collect data from having a thumbs up or thumbs down directly indicating that they like or dislike a particular item [4]. These events are mapped directly in the ratings matrix, and explain directly how a user feels about an item. The next section explains the design decisions regarding the explicit feedback we will collect in the WOTM application.

Boolean Ratings vs Ternary Ratings vs Likert Scale Ratings

Although prediction-accuracy is important, it is not the only factor that a recommender should focus on and acts as one facet in wide range of facets [23]. Martin explains that the goal of a recommender system is to improve user experience however, designing a recommender system to fit a specific application remains a challenge, and recommender system ratings should be based on the user experience [23]. By focusing on user experience, users will be able to easily rate food dishes, in turn, leading to the system collecting more information for the recommender system from ease of use. For this reason, a simple model allowing users to easily rate a dish such as a binary rating (Like/Neutral) or a ternary rating (like/neutral/dislike) is preferred over Likert scale ratings. This provides simplicity, but means recommendations will not be as accurate as Likert scale ratings such as 1-5 stars or 1-10 stars. By easing the user experience for rating food dishes it can increase data collection at the expense of accuracy. In fact, Sen et al. found that users provided more ratings that had options to "like" or "dislike" than users with only one rating option [**?** 31].

Using Likert scaled ratings mean that we learn more about the user preferences because of the scaled factor indicating how much the user likes a dish or not. This means that model based CF techniques are able to learn what the user likes and dislikes faster leading to more accurate recommendations. However, it is difficult to see how good a dish is by using a Likert scale system such as a 1-5 star rating since the ratings will be skewed. For example, if a dish had a 5 star rating, it usually means that only 1 or 2 people have rated that dish. Ratings such as 3.5 stars may mean that it is a good dish, but from the way it is displayed, may seem otherwise. Cosley et al. explains that by displaying what other users think of an item, the active user tends be influenced by the opinions of others, leading to bias ratings [24]. An example would be a user rating a dish higher than they would normally because the food dish has an average of 4.5 stars. This can lead to inaccurate recommendations for the active user in the long term, because users may be influenced by others opinion [24]. Cosley et al. argues that the way ratings are collected, and displayed influence how others will rate the dish. Considerations like this have to be taken into account to understand how the user will perceive and interact with the recommender system.

A trade-off to consider is whether or not accurate recommendations are more important than the user experience. Since WOTM aims at being a mobile application, the limitations are the small form factor that mobile phone screens have. With a Likert scale rating system, the user has to be shown these possible options in order to rate a dish. This can take up additional space that is not needed on small screens. Because of this, having a 1-5 star rating system may degrade the user experience as opposed to a simple like/dislike rating system.

For a mobile application, prediction-accuracy of the recommender system may not matter a great deal. For instance, a recommender system could predict two dishes the user may like based on previous rating patterns. The first dish is predicted with a 90% prediction accuracy, and the second dish is predicted with a 70% prediction accuracy that the user will like these dishes. But is the difference in prediction accuracy important if the user likes both dishes? As long as the recommender system has provided the user with dishes they like, the accuracy between those predicted dishes do not drastically matter. In addition, dishes with higher predicted accuracy may seem like obvious choices that the user may have already tried, whereas dishes with a lower predicted accuracy may lead to less obvious dishes that they may like, but have not tried yet. A recommender system using boolean or ternary values would eventually reach prediction accuracy similar to using likert scale ratings. Although this will happen in the long term, new ratings will most likely lead to less accurate predictions that join the system in the short term until more ratings are collected.

Foursquare is an application that asks users to rate items according to a series of questions. These questions consist of ”What do you like about this place?”, ”What is this place known for?” and so on. From these questions they are able infer a particular rating for the item, as well as collect data from users to give more accurate recommendations. This rating system may risk users not rating the items because of the long list of questions it asks. [23] explains that part of the challenge is to design interfaces ”that give users control over the recommendation process without overwhelming the user or rendering the tool too complicated for novice users.”

For these reasons, we found that simple like/dislike events would best suit the collection of data for the recommender system because of the simplified structure which increases the user experience. The recommender system can be extended to take in additional events that may portray additional information such as a ”want” indicating that a user ”wants” a dish but has not yet tried it before. However, caution must be taken as more events will affect the user experience of the application, but may lead to more accurate recommendations.

3.2.4 Implicit Feedback

When explicit feedback is not available, implicit data can be used to infer preferences from users [4]. Implicit feedback is referred to as an indirect reflection of someones interest in an item. Implicit feedback can be from observing user behaviour. This could include click through data, browsing history, the way users react to certain events, search patterns, and so on [4]. Implicit feedback can be collected to increase the accuracy of recommendations to users by being combined with explicit feedback, or can fill in the ratings matrix when explicit events are not available, alleviating the ’cold start’ problem as it makes the matrix more dense. The next section explains design decisions regarding the implicit feedback in WOTM.

Additional Events

Netflix and other sites such as Amazon.com [4, 2] use implicit feedback such as views and purchase history in their recommender systems. These events indicate some form of interest in the user, however are less practical to apply for a mobile application. For example, on a website, many products are able to be shown to the user. When a user selects a product it will indicate some form of implicit feedback such as the user is interested in that product. With the WOTM mobile application, it can be diffiuclt showing a range of food dishes to users because of the small screen size. This can make it difficult to collect additional implicit feedback. As well as this, purchase history and similar events are not practical because users are unlikely to indicate that they have purchased a dish after they have tried it. Explicit events such as like and dislike already infer that they have tried the dish, which make purchase history redundant. The implicit event of commenting on a dish may provide valuable information, perhaps commenting on a dish means that there is a strong interest or disinterest a user has about a specific dish. However, this is impractical because we do not know if the comment is good or bad without using extraction techniques.

A consideration to think about is how to collect feedback on how the user feels about the recommendations produced by the recommender system. This can be used to gather more events, to make predictions more accurate. For instance, a user can indicate that they liked/disliked the recommendation that was shown to them, or they could skip it altogether meaning that they they do not have an opinion about it. Although the flaw in this method is that recommendations may be dishes the user has not tried yet. Therefore, they may keep skipping through recommendations and provide no valid feedback to them. Although this may happen, an assumption is that user ″Like″ events may be used for a dish that the user has not yet tried, but is interested in.

For these reasons, we do not use any implicit events in our recommendation system but this may change in future.

3.3 Implementation

This section explains what has been implemented according to the design decisions in the previous section.

3.3.1 Latent Factor Models

PredictionIO [29] provides many template engines, some of which use CF algorithms to make recommendations to users. In particular, there is a template engine that uses Singular Value Decomposition, learning user preferences based on previous rating patterns. This engine is called the ″E-Commerce Recommendation Engine″, and will be built upon to suit our use case. The following section explains the E-Commerce Recommendation Engine and the changes that have been made to it for this project.

E-Commerce Recommendation Engine

The E-Commerce Recommendation Engine is written in the programming language Scala and is made to provide personalised recommendations for e-commerce applications. This engine uses Singular Value Decomposition which is a model-based CF technique. This means that the model trains on the users rating patterns to learn about recommendations that the users may be interested in. Training occurs offline. Default events in the engine are view events and purchase events. The engine also comes with the following out of the box functionality [29].

1. Exclude out-of-stock items
2. Provide recommendation to new users who sign up after the model is trained
3. Recommend unseen items only (configurable)
4. Recommend popular items if no information about the user is available

Since ″view″ and ″purchase″ ratings are implicit events and not explicit events such as Likert scaled ratings from values 1-5, the implicit events have to be represented differently in the ratings matrix. Implicit events are represented in the ratings matrix by the combination of binary preferences and the confidence values of these binary preferences being true instead. An implicit event such as ″view″ or ″purchase″ event maps to a binary preference value of 1. Missing events map to a binary preference value of 0.

A binary preference value of 1 indicates that the user likes the item, and a binary preference of 0 indicates no preference for the user. Additionally, implicit events also have a confidence value associated with it. These confidence values represent the confidence levels of the binary preference values being true. In this case, preference values for the "view" and "purchase" events correlate to the value of 1. Multiple identical events such as a user viewing the same item multiple times correlate to higher confidence levels since the confidence level is an aggregation of preference values. This means that there is a higher confidence level that the binary preference value is true, in this case, that the user will like the item because the user viewed or purchased the same item multiple times, recommendations taking this into account.

Singular Value Decomposition is then used with the same steps in equation 2, except with a different minimization equation that considers the implicit events. This minimization equation is for implicit events which is the following [32].

$$min_{q*,p*} \sum_{(u,i)\in K} c_{ui}(b_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \quad (3)$$

In this equation, c_{ui} is the confidence level and b_{ui} is the binary preference value.

Modifications to the Engine

Using this template, we modified this engine to take into account "Like" and "Dislike" events, removing the view and purchase events. We modified the CF algorithm in the engine to consider a "Like" event as positive result, giving it the preference value of a 1. In contrast, we made the algorithm in the engine consider a "Dislike" event to be a negative result, giving it the preference value of a -1. Since users may be able change their minds about liking or disliking a dish, we modified the CF algorithm to only take into account the most recent like/dislike event that occurs from the user, if there are multiple identical like/dislike events on a dish from that user. Filtering in this recommendation system was also added. Users are now able to filter recommendations based on their preferences such as their meat type, their cuisine type, and their food type. As well as this, users can see recommendations that are within their price range, if specified.

If dishes are no longer available, then we are able to send a query to the recommendation engine to tell it that the dish is unavailable. Another feature is that we are also able to send a query to the recommendation engine to say that a user has already seen that dish, and not to recommend that seen dish anymore. In this way, the user only sees new dishes that they have not been recommended yet.

This model recommends dishes to users as soon as they have liked a dish. These recommendations are also based on the ratings of other users rating patterns, which means that the ratings matrix should be dense. If the model cannot learn what the user likes, or the user has not rated anything yet, then it defaults to recommending the most popular dishes to the users. The advantage of this is that users get to see the trending dishes that other users prefer, however the disadvantage is that it may create bias results in the system because popular dishes will only be seen by new users. This means that new users will only rate the popular dishes, causing problems in the recommender engine. To extend this engine, we need to only recommend dishes to users after they have rated x amount of dishes. Random dishes should be shown until they ahve rated x amount of dishes.

Chapter 4

Future Plan & Request for Feedback

4.1 Future Stages

The main components remaining for this project include the implementation, user study, evaluation, and the final report write-up. Currently we a SVD CF method working. Pending implementation sections include implementing neighbourhood CF methods. On the completion of these tasks, we must collect user information to test these methods. We are then able to run evaluation methods on real user data to obtain results. The main consideration is whether or not we want to test the performance or the prediction accuracy of the recommendation systems. Some feedback would be appreciated in terms of this.

4.2 Evaluation Methods

Since we are collecting like/dislike events from users, we would like a way to evaluate these methods. We would potentially like to evaluate these methods in terms of speed of recommendations, and prediction accuracy (depending on above).

In terms of prediction accuracy, we are thinking of asking users to label food dishes. They will label the food dishes that they have tried before, labelling whether they liked the food dish, or whether they disliked the food dish. They will label as many of the food dishes as they can. We will then use 75% of this data to train our model on, learning what they like. We will then use the remaining 25% of the labelled data as a test set, using the recommender system to predict whether the user will like the remaining 25% of the data. The prediction-accuracy will be based on how many recommendations that the system got right.

In terms of performance, we are not sure on what to do to test this.

4.3 Other collaborative filtering methods

Collaborative filtering has a range of different techniques. In the time frame given, it is difficult to test every method. In addition, parameters in collaborative filtering techniques also need to be tuned according to the data collected. In future, we plan to examine the neighbourhood CF techniques, comparing them with the SVD approach.

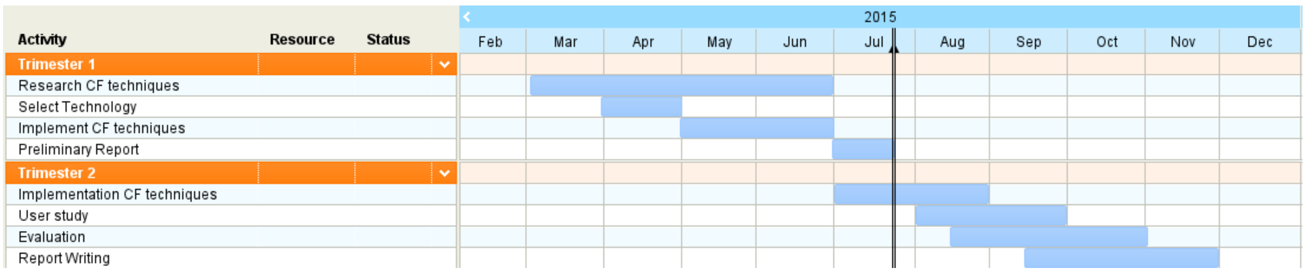


Figure 4.1: Full project Gantt Chart

4.4 Proposed Timeline

We would like some feedback as to whether this project is feasible or not. Feedback regarding the evaluation plan would be kindly appreciated as well.

Bibliography

[1] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[2] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan, "Collaborative filtering recommender systems." Now Publishers Inc., 2011, vol. 4, no. 2, pp. 81–173.

[3] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[4] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[5] J. Lee, M. Sun, and G. Lebanon, "A comparative study of collaborative filtering algorithms," *arXiv preprint arXiv:1205.3193*, 2012.

[6] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in action*. Manning Shelter Island, 2011.

[7] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender systems handbook*. Springer, 2011, pp. 145–186.

[8] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[9] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

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

[11] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 1994, pp. 175–186.

[12] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE, 2007, pp. 43–52.

[13] B. Sarwar, G. Karypis, J. Konstan, and Riedl, "Application of dimensionality reduction in recommender system-a case study," DTIC Document, Tech. Rep., 2000.

[14] G. Karypis, "Evaluation of item-based top-n recommendation algorithms," in *Proceedings of the tenth international conference on Information and knowledge management*. ACM, 2001, pp. 247–254.

[15] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.

[16] L. H. Ungar and D. P. Foster, "Clustering methods for collaborative filtering," in *AAAI workshop on recommendation systems*, vol. 1, 1998, pp. 114–129.

[17] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 89–115, 2004.

[18] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 6, pp. 734–749, 2005.

[19] P. Melville, R. J. Mooney, and R. Nagarajan, "Content-boosted collaborative filtering for improved recommendations," in *AAAI/IAAI*, 2002, pp. 187–192.

[20] D. Y. Pavlov and D. M. Pennock, "A maximum entropy approach to collaborative filtering in dynamic, sparse, high-dimensional domains," in *Advances in neural information processing systems*, 2002, pp. 1441–1448.

[21] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 426–434.

[22] Y. Koren, "The bellkor solution to the netflix grand prize," *Netflix prize documentation*, vol. 81, 2009.

[23] F. J. Martin, "Recsys' 09 industrial keynote: top 10 lessons learned developing deploying and operating real-world recommender systems," in *Proceedings of the third ACM conference on Recommender systems*. ACM, 2009, pp. 1–2.

[24] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl, "Is seeing believing?: how recommender system interfaces affect users' opinions," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2003, pp. 585–592.

[25] E. Bernhardsson. Spotify/annoy. [Online]. Available: https://github.com/spotify/annoy

[26] D. Celis. Recommendable. [Online]. Available: https://github.com/davidcelis/recommendable

[27] Sourceforge. easyrec. [Online]. Available: http://easyrec.org/

[28] GroupLens. Lenskit. [Online]. Available: http://lenskit.org/

[29] PredictionIO. Predictionio. [Online]. Available: https://prediction.io/

[30] S. Chan, T. Stone, K. P. Szeto, and K. H. Chan, "Predictionio: a distributed machine learning server for practical software development," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2013, pp. 2493–2496.

[31] S. Sen, F. M. Harper, A. LaPitz, and J. Riedl, "The quest for quality tags," in *Proceedings of the 2007 international ACM conference on Supporting group work*. ACM, 2007, pp. 361–370.

[32] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 263–272.