Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FooDroid: A Food Recommendation App for University Canteens

Semester Thesis

Marian Runo
runom@ee.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Tobias Langner, Samuel Welten
Prof. Dr. Roger Wattenhofer

June 21, 2011

Acknowledgements

I thank my advisors Tobias Langner and Samuel Welten for their helpful advices and constructive discussions.

Abstract

The goal of this semester thesis was to develop a menu recommendation system for the canteens of ETH Zurich (and University of Zurich). Existing applications mainly present the menus for the day in lists but cannot tell which menu a user would prefer. Browsing long lists to find a favorite dish however is tedious. The system should be able to recommend available menus based on the preferences of the user. Mobility was also a key feature to consider because users would most likely not have access to a computer on their way to the canteen, however mobile phones are almost ubiquitous. The final implementation consists of three parts: A crawler which downloads the necessary data and adds it to a database; a server which retrieves menu data from the database and computes ratings; and a client running on an Android-based mobile platform which presents the recommended menu to the user in a simple and convenient way.

Keywords: android, app, mensa, canteen, menu, recommendation, collaborative filtering, ETH, university, Zurich

Contents

Chapter 1

Introduction

With many offerings always comes a selection process which can become quite cumbersome very fast if many different offers have to be compared to make an optimal choice. One thing where tastes could not vary more widely is food. Simply rating a canteen or restaurant can only give a very generalized assessment about the quality of the different menus served. Some menus might be favored by some customers but not others and depending on who gives more ratings, the recommendation accuracy suffers. A more personalized recommendation system which operates on the preferences of individuals – or small groups – is hence desireable. Addtionally, the location of the customer and a canteen or restaurant comes into play if only a limited amount of time can be spent on lunch. A customer should not be forced to walk across town. Rather, a combination of good menus and relative closeness needs to be considered in the recommendation process.

In particular, the number of canteens around the ETH Zurich and University of Zurich is big and often they are located far apart (Zentrum, Science City, Irchel). Often people are not willed to invest much time in searching for preferred menus on the canteens websites before lunch – or they forget to check altogether. Even if a customer is willed to browse the various canteens' websites for menu lists though, the sole descriptions oftentimes do not provide great insight into the actual edibility of a menu. *FooDroid* aims to solve these problems by providing a unified platform to display menus of various canteens on Android based mobile phones. Users can browse through the menus offered today in various canteens, filter meals according to their taste and rate them for latecomers to benefit from the reviews of their fellow colleagues.

1.1 Existing Applications

Before *FooDroid*, students had to rely mostly on websites provided by the canteens themselves in order to gather menu information. However, there exists a simple application for Android-based mobile phones called *StudiFood* which was

programmed by fellow student Mirko Mikulic. The application lists the various menus according to their corresponding canteen and provides a distance measure. There is, however, no rating system in place, hence the application can be understood rather as a more comfortable way to access the data provided on the various canteens' websites than a real recommendation solution. The same can be said about *ETHZ Mensa - Uni Zürich Mensa* for iOS (iPhone) which provides, as stated in the description, "Nothing more, nothing less" than a simple list. *FooDroid* picks up the idea of *StudiFood* and others but tries to extend it in order to provide an additional accurate recommendation system to avoid tedious browsing of dozens of menus.

1.2 Related Work

Even though recommendation systems aren't exactly a new idea they experienced a recent gain in popularity. The work done on GroupLens [4] for instance, which this work bases a part of the recommendation procedure on, was developed in 1994 for rating prediction on Usenet newsgroup entries. Fortunately many rating algorithms can be used independently of the actual thing to be rated. Because these systems rely on user input, originally only few places which actually supported user feedback could employ such systems. Most of the internet of the past however was a static assembly of webpages with no possibility for users to interact. With the advent of Web 2.0 which incorporates the actions of users much more, these recommendation system experienced a big gain in popularity.

There has been done considerable work for canteen or restaurant-based recommendation systems over the years. *REJA* [2] is such a restaurant-based recommendation system which uses a hybrid recommendation algorithm based on collaborative filtering as well as knowledge-based filtering. *REJA* further incorporates a geographic module that provides information referred by Google maps. However, this system is a web-application which can be a hassle to use on mobile devices. Another interesting approach is taken by Moon-He et. al. [3]. They try to recommend restaurants to groups of people which can have widely differing tastes. Taking into consideration whole groups of people required the use of a complex AHP (Analytic Hierarchy Process) which works on multiple criteria for its decision making. The client application however only runs on Windows-based mobile devices of which the market penetration, compared to the desktop segment, is rather low.

There are only few recommendation systems which specifically deal with menus though and are often tailored to users under dietary constraints [1]. The goal is to bring a recommendation system to an up-to-date platform, in this case Android, and provide highly personalized menu recommendation.

Chapter 2

System Architecture

First of all an overview on the system architecture is provided to establish the basic structure and basic workings of the whole. Then a more detailed view on every component is provided, explaining in greater detail how the individual components interact and what their exact purpose is.

2.1 Overview

Simple data fetching would usually only require a client application which can fetch menu data from the various canteens' own websites and display them in a structured manner. *FooDroid* however wants to provide the students with a mean to rate menus he or she ate and recommend menus based on these ratings in the future. This is more complicated than the existing approach by *StudiFood* and hence requires a more complex system of interacting software. The *FooDroid* system (see figure 2.1) hence consists of three major parts:

- a data crawler, for fetching the data from canteens' websites
- a data server, for providing data for the client application
- and the client application itself on the Android phone

The whole system is backed by a MySQL database wherein the crawled data as well as ratings for each menu and user are stored. The following sections will explain each part of the system and their inner workings in greater detail.

2.2 Data Crawler

The data crawler is the most important part of the whole since it crawls the different canteens' websites for the menus, parses the data according to the system's needs and adds them to the database. The data can then be easily accessed and

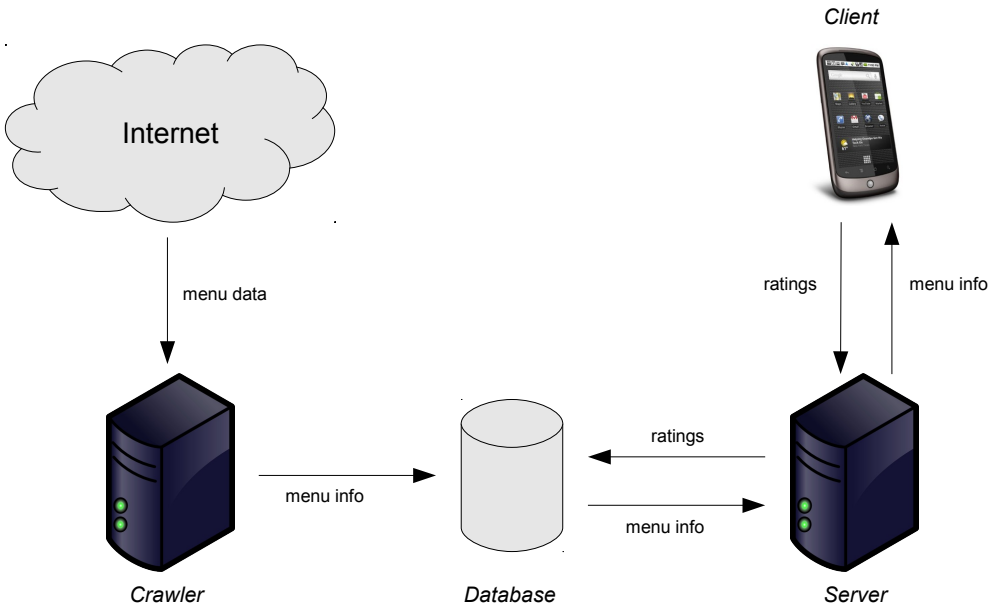


Figure 2.1: System Overview

is returned in human readable form as an xml file. There is no restriction in place concerning the access to the data nor is there a dependence on a certain client, i.e. other applications could potentially access the data as well as add ratings for menus if it would be decided to make the database public access. The crawler's work itself can be divided into the following steps:

1. Downloading the menu data provided by the canteens
2. Extracting informations such as menu name, menu description and price
3. Extracting token words and ingredients used from the menu's description
4. Adding all the newly gained information for each menu to the database

The downloaded data is parsed into a class called *MenuItem* (see table 2.1) and holds all the necessary variables which will be written to the database. The process of writing the data to the database begins with checking if a menu with identical variables is already present for today. If true, the writing process is skipped. Otherwise an entry in *MenuItems* is created. See figure A.1 for the database's structure. The database returns the *menuitem_id* of the newly created row and the crawler then proceeds to add the new ingredients. Usually, there will be no additions because the dictionary is static, i.e. updated by hand. However, if new entries are made in the dictionary, these entries are automatically added to the respective tables if they are found in a menu. The ingredients have to be

Variable	Description
String mensa	The name of the canteen the menu belongs to
Boolean closed	Certain keywords trigger this varbiable and if true, the menu is not added to the database
String menuName	The name of the menu, e.g. "Menu 1"
String type	Specifies if the menu is served during lunch or dinner
String description	The description of the menu as extracted directly from the downloaded data
String validDate	The date on which the menu is served
String[] tags	Contains the extracted tags of the menu, e.g. "vegetarian"
String[] ingredients	Contains the extracted ingredients of the menu
String[] tokens	Contains the exploded menu description (for future use)

Table 2.1: MenuItem variables

connected to the menu which is done by adding pairs of *menuitems_id* and *ingredient_id* to the *MenuItem2Ingredient* table. The procedure for the ingredients is identically executed for tags and tokens using the tables *Tags*/*MenuItem2Tag* and *Description_Tokens*/*MenuItem2Token* respectively.

There is no restriction on the number of canteens or the form of the data they provide. The crawler's parser class is abstracted in order to insure easy addition of canteens in the future. For the time being, parsers for two major canteen providers, ETH Zurich and University Zurich, have been implemented.

2.2.1 ETHZ Parser

The ETH Zurich provides data on their menus in the form of html tables as well as RSS feeds (XML). For simplicities sake, the easier xml data format was chosen. The parser expects a list of URLs from where the data should be downloaded. This approach was chosen to insure high flexibility if a URL changes or a need arises to exclude certain canteens altogether (for instance cafeterias with non-changing offers). This way menus of 14 canteens all around the ETH Zurich can be included into the database.

2.2.2 UZH Parser

Data on menus for the University Zurich is distributed in the form of pdf files. Pdf files are much more challenging to parse than a simple xml file because one has to rely on whitespace characters for its structure. For the time being parsers for the two closest and hence most interesting canteens, Zentrum für Zahnmedizin and Plattenstrasse, were implemented. This can be easily extended in the future

to include other canteens such as the canteens situated in the main building of the university such as Mensa A/B as well as Mensa Irchel.

2.3 Server

The client gets his data from the server which has direct access to the database and consists of several PHP scripts. The client can ask for specific data only considering certain canteens or ingredients which is then retrieved from the database by the server and returned to the client formatted as an xml file. It is also the job of the server to calculate all kinds of ratings.

While the base rating and ingredient ratings are calculated offline, some ratings such as the user rating and the correlation between users are calculated on demand. This could potentially lead to scalability issues in the future though. For many users the computation time of the user-based collaborative filtering grows quadratically. In order to circumvent this problem, correlation coefficients could be calculated offline and stored in a database every few hours to incorporate new ratings. This way the calculation is only done a few times over the course of a day and not every time a user requests the menus.

2.3.1 Database

Menu data as well as user data and ratings are saved in a MySQL database. The database consists of several tables each containing specific data. Table 2.2 shows a listing of the used tables and their purpose while figure A.1 shows the database's actual structure. The database is highly normalized to minimize redundancy. For the ingredients, tags and tokens an intermediate many-to-many mapping table was employed. This tag database structure is based on the Toxi tagging scheme which is also used by Wordpress.

2.4 Client

The *FooDroid* client retrieves menu info from the server and displays it in a convenient manner for the user. Upon starting the app, the user is presented with the menu of the day (see fig. 2.2(a)). This menu is chosen by the highest rating which is a combination of three separate ratings, ingredient-based, user and user-based collaborative filtering. Further options can be accessed by pressing the phone's settings button. There the user is presented with several possible actions:

- **Update** retrieves updated menu info and recommendation with updated filters from the server

Name	Description
Mensas	Informations about canteens such as id, name, location
Periods	Times for lunch, dinner in order to present menus according to their serving period
Ratings	Ratings for each menu with related user id
MenuItems	Data on menus such as id, name, description, base rating
Description_Tokens	Token words extracted from the descriptions of menus
Ingredients	Ingredients extracted from the descriptions of menus
IngredRatings	Ratings for each ingredient, calculated from user ratings
Tags	Tag words such as vegetarian etc.
MenuItem2Ingredient	Maps ingredients to menus
MenuItem2Tag	Maps tags to menus
MenuItem2Token	Maps token words to menus
Users	User data such as id, hashed IMEI and joining date

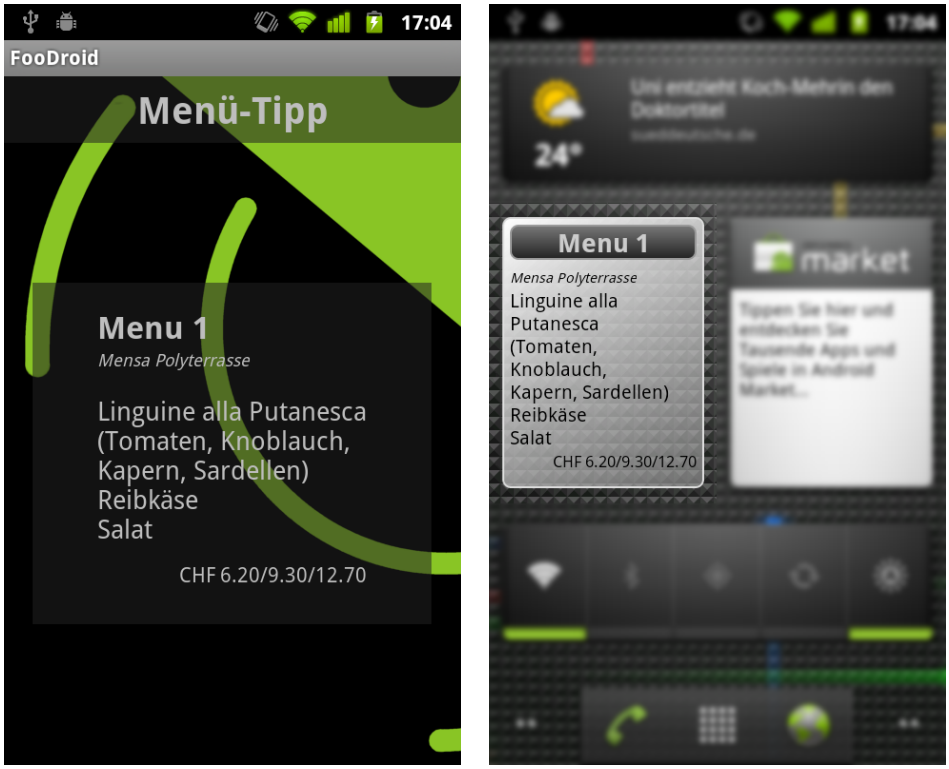
Table 2.2: Database tables

- **Browse** the user can browse all menus sorted by canteens
- **Settings** ingredient filters and canteen filters can be set as well as a reminder option
- **About** about the application

As mentioned, if the user wants to check for further menus, it is possible to browse all menus for the day sorted by canteens (fig. 2.3(a)). While browsing the user can also rate menus (fig. 2.3(b)) consumed today so that other users who have not eaten yet may profit from the newly gained knowledge. The user can rate a menu by adding stars by dragging the filling from 0 to 5 stars. In order to prevent erroneous ratings caused by inaccuracies which surface by handling a touch interface, the user is prompted to confirm the rating which was just given. If the rating was indeed erroneous, the user can then cancel the rating submission and re-submit the intended rating.

User's preferences can be set under "Settings" where certain ingredients and canteens can be excluded. Excluded ingredients and canteens then won't be considered for the menu of the day and won't show up while browsing. The client also includes a so called "AppWidget" (fig. 2.2(b)) which can be placed on the phones home screen. This way the user has even easier access to the recommended menu. The widget updates itself every few hours if the phone is active. Sometimes the user does not use the phone for a prolonged period of

time and the widget is not updated to save battery power. In such a case the user can simply touch the widget and the menu info is automatically updated.

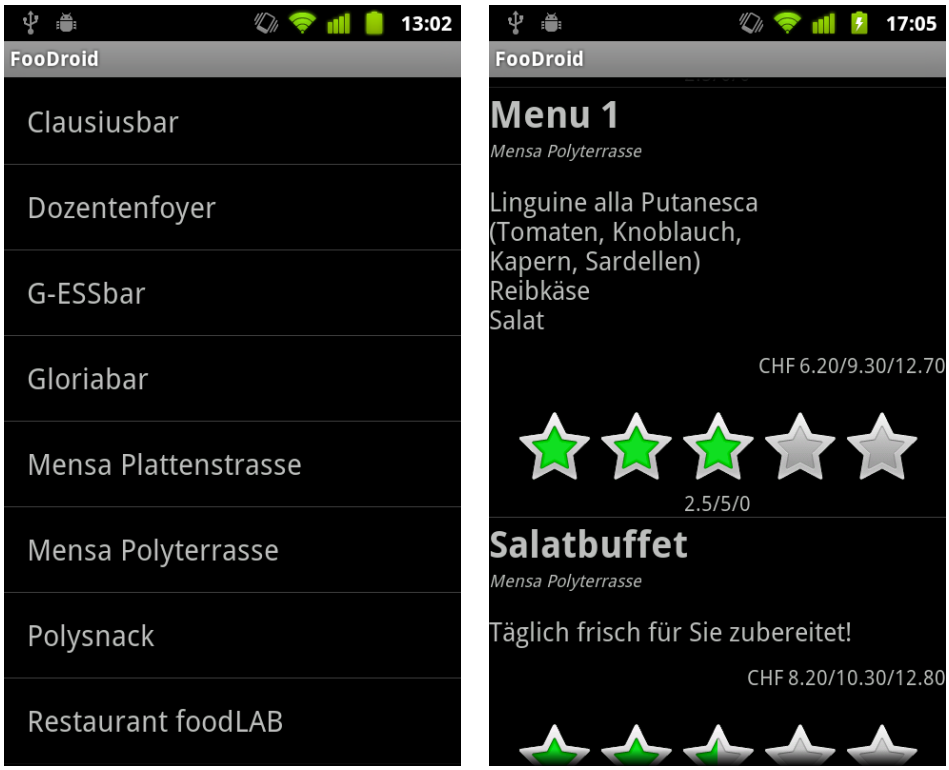


(a) (b)

Figure 2.2: Client UI: a) Home screen which shows the recommended menu for today, b) AppWidget on the phones "desktop" recommending the menu for today directly without starting the application

Getting as many ratings as possible can be an issue because many users simply cannot be bothered to rate. However, in order to insure high prediction quality as many ratings as possible need to be submitted to the system. Only then can sufficiently accurate recommendations be provided. In order to motivate people to rate menus, a reminder was implemented. If the user choses to be reminded (under settings), half an hour after the application has been started, a notification will be issued in order to remind the user to vote. Experience has shown that while there is an interest in checking the recommended menu, users often forget to rate after eating because the application is not needed anymore and hence the focus of attention shifts to other matters. If the user rates a menu before the notification is issued, the pending notification is canceled and the user is not reminded this day anymore. Further, there is no additional notification issued after the user has been notified one time even if the application is launched several more times. This was done in order to reduce the number of interruptions

and trouble, and to prevent the user from unchecking the reminder option and forgetting to rate altogether.



(a) (b)

Figure 2.3: Client UI: a) Canteen listing which is dynamically loaded from the server, once the user chooses a canteen by tapping, he or she is greated with the b) Menu listing for the specific canteen with rating indicators; The three numbers below the rating stars show the individual ratings, ingredient-based, user and collaborative filtering

Chapter 3

Menu Recommendation

In order to recommend a menu some kind of rateable variable has to be introduced since the crawled data has no such attributes by itself. Therefore the system has to assign a rating to each menu. An overall rating for a certain menu is calculated by combining several separate ratings into one. Seperate ratings have to be considered because one rating rarely covers all aspects of what it takes for a menu to be good or bad. There are further limitations like the availability of up-to-date user ratings whose lack causes certain ratings to be non-usable but which, on the other hand, have far better prediction accuracy than any other rating if up-to-date ratings are indeed available. The ratings also vary by the direct influence of the users.

At this time three separate ratings are implemented, though if need be, additional ratings can be added easily in the future. A sensible method of combining the ratings had to be developed reflecting the differing predictive qualities of each rating. Since the optimal combination of ratings is still an active area of research, a basic combination method was chosen which might not be optimal but somewhat reflects the significance of the separate ratings.

3.1 Ingredient-Based Rating

In order to have a first assessment of a menu, the ingredient-based rating is calculated. This rating is calculated by taking into consideration the ratings of individual ingredients of a menu hence does not rely primarily on recent user ratings. This is important because at the start of a period or day obviously no user had the opportunity to rate a menu yet. Even under these circumstances though, a recommendation has to be provided. An additional table aids in this endeavour by providing an averaged ingredient rating which was computed from all rated menus containing this ingredient (see figure 3.1 for how the table is built). The algorithm checks if an ingredient found in the description of the menu to be rated is listed in the *IngredRatings* table which holds all the ingredient ratings. It then fetches the ratings of all ingredients present in the menu to be

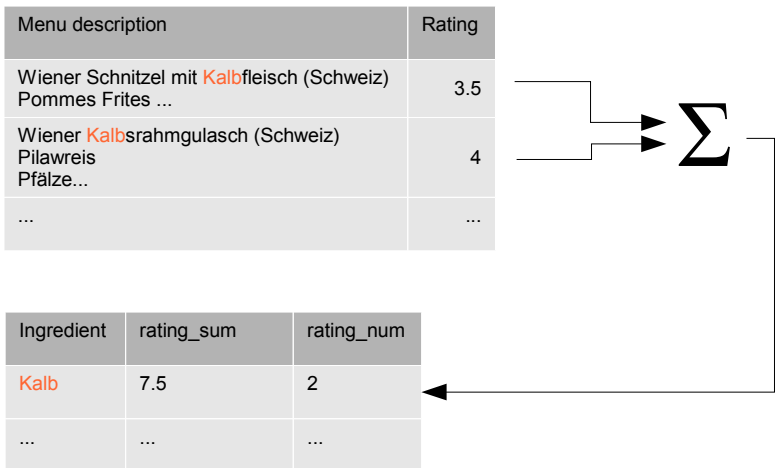


Figure 3.1: Building the *IngredRatings* table

rated and adds them up. Finally the average is calculated by dividing the sum by the number of rated ingredients. The rating can then be written as:

$$R_B = \frac{\sum_i \frac{rating\_sum_i}{rating\_num_i}}{n} \tag{3.1}$$

where $rating\_sum_i$ and $rating\_num_i$ are the summed ratings and number of ratings respectively for a particular ingredient i of the n ingredients the menu contains. These values are, as just mentioned, stored and retrieved from the database. In the case that no ratings have been given for the ingredients in the menu to be rated, a neutral rating of 2.5 is given on a scale of 0 (bad) to 5 (excellent). This base rating relies on past data and works with keywords from a dictionary hence the results can vary significantly. Even though it is a very rough rating, the base rating can provide a good guide if no users have rated a particular menu today, i.e. no recent user feedback was received.

3.2 User Rating

The user rating reflects the opinion of users about a particular menu and is the second most important rating and a way for the user to directly communicate with the system. Where the ingredient-based rating is a very crude first guess, the user rating is regarded as a more accurate idea. Of course, the opinion of one user is not enough to give a definite recommendation hence the more user ratings are given, the better the whole menu rating works. In section 3.4 the process of combining the seperate ratings and how the system avoids high impact of single

high or low user ratings will be explained in more detail. Every user can give a rating for a menu between 0 (bad) and 5 (excellent). The user rating for a particular menu is then simply the average of all ratings for this menu:

$$R_{U,m} = \frac{\sum_{i=1}^{n} R_{m,i}}{n} \tag{3.2}$$

where $R_{U,m}$ is the current user rating for menu m and $R_{m,i}$ is the i^{th} rating out of n ratings for menu m. The user rating is used for the calculation of the ingredient ratings as well as for the computation of the correlation coefficients and later the user-based collaborative filtering. Because of the fact that the other ratings rely on the ratings given by users, it is important to have as many user ratings in the database as possible in order to increase prediction quality.

3.3 User-based Collaborative Filtering

Collaborative filtering entertains the idea that people who agreed in the past will most likely agree in the future. Simply put, if someone rated menus very close to how I did, it's very likely that if someone liked a menu I haven't rated yet, I will like it too. Collaborative filtering is generally accepted to be one of the most effective mechanisms for producing high quality predictions [5] and is treated as such. The problem however, is that a recent rating for the menu is needed in order to give a prediction, meaning that someone has to rate a menu before the collaborative filtering can be used to recommended a menu to another user. The lack of prediction capability of this collaborative filtering method until ratings have been given therefore requires a sensible method of combining the different ratings such that non-significant ratings can be suppressed.

The collaborative filtering algorithm used in *FooDroid* is a memory-based type and was employed by GroupLens [4]. It uses weighted averages of all ratings for a certain menu. For this rating, first all Pearson correlation coefficients between the user and all other users have to be calculated. The Pearson correlation coefficient between two users, Ken and Lee, can be calculated as follows:

$$r_{Ken,Lee} = \frac{E[(K - \mu_{Ken})(L - \mu_{Lee})]}{\sigma_{Ken} \cdot \sigma_{Lee}} \tag{3.3}$$

where σ is the standard deviation, μ is the average of all ratings for this user and E is the expected value of the differences of all ratings of Ken K and Lee L and their respective averages. It is assumed that the probabilities of all ratings are uniformly distributed. The correlation coefficient can hence also be written as:

$$r_{Ken,Lee} = \frac{\frac{\sum_{i=1}^{n} (K_i - \mu_{Ken})(L_i - \mu_{Lee})}{n}}{\sigma_{Ken} \cdot \sigma_{Lee}} \tag{3.4}$$

where n is the number of the common rated menus of the two users. The final rating can now be calculated with the weighted average of all users:

$$R_{CF_{Ken},m} = \mu_{Ken} + \frac{\sum_{J \in users}(J_m - \mu_J) \cdot r_{Ken,J}}{\sum_{J \in users} |r_{Ken,J}|} \tag{3.5}$$

where m is the menu.

3.4 Rating Combination

Since there are three ratings with varying degrees of accuracy and availability, one has to chose a way to combine all the ratings into one final rating. At the start of the day there are no user ratings present hence the final rating relies heavily on the base rating first. The more people rate a menu, the less significant the base rating becomes. The process of combining the three separate ratings into one looks as follows. First of all the base rating and current user rating are combined. The weight function used looks as follows:

$$w(x) = \begin{cases} 0 & \text{if } x < 0 \\ \frac{1}{5} \cdot x & \text{if } x \geq 0 \\ 1 & \text{if } x > 5 \end{cases} \tag{3.6}$$

where x equals the number of ratings for a menu. It was decided that after five

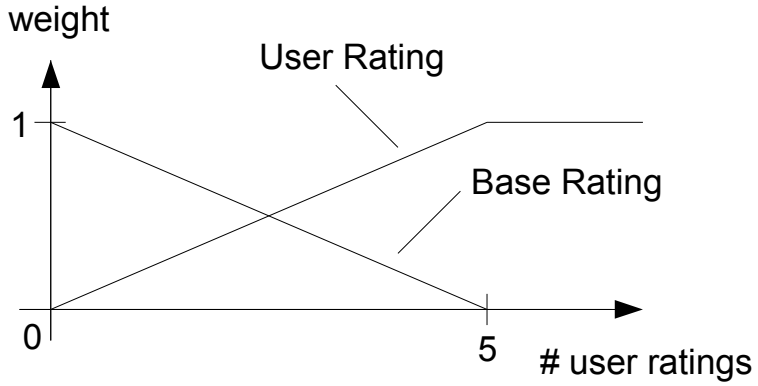


Figure 3.2: Weight Function

users have rated a menu, the base rating shall have no further influence on the overall rating. This number is arbitrary and can be adjusted if need be. Both ratings are then combined into an intermediate rating:

$$R_{int,m} = (1 - w) \cdot R_B + w \cdot R_{U,m} \tag{3.7}$$

In order to get the final rating, $R_{int,m}$ and the rating based on collaborative filtering are simply averaged, although a different approach like for the base and user rating might be possible:

$$R_{final,m} = \frac{R_{int,m} + R_{CF_{User},m}}{2} \tag{3.8}$$

The client then chooses the recommended menu according to the highest rating. For now, only one recommended menu is shown although it might be beneficial to present the user a short list of the three or five highest rated menus.

Chapter 4

Conclusion

Recommending menus can be a demanding task because simple ingredient based comparison might produce misleading predictions. Therefore it was decided that the opinion of the users has more weight in the recommendation process than automated ratings. A system was developed which can list menus without constant user input but can also improve predictions by taking into account the ratings of users. Preferences are expressed by rating presented menus but also by specifically including or excluding liked and disliked ingredients respectively.

4.1 Future Work

There are a few things which had to be cut in order to fit into the time frame of a semester thesis. For one, the system could not be evaluated in this short time. To do that a large number of ratings and users are needed but were not available because the whole system was written from scratch and there was no ready for release version until late in the semester. However it is very important to check if the system produces better predictions than a simple random approach. Further, the question remains how different combinations of ratings compare and what different combination procedures could improve the prediction. The three ratings and their combination in this work were chosen as an educated guess and there is no definite way of telling if this choice was any good. Other collaborative filtering methods and types such as model-based collaborative filtering could be used. Location-based filtering is another feature which could be implemented so that only menus of canteens in a user-definable radius or area are considered. This was skipped in this iteration because of delay considerations; the location detection needs some time to lock unto a new location but that would cause unwanted delays in the retrieval of the menus. A new approach would be needed to incorporate the location without noticeable delays. A new feature with a big impact might be group based recommendation like proposed in [3] which would try to recommend a menu or canteen to a group of users. Of course, most often not all users prefer the same menus so a simple majority vote would be

insufficient. Bringing the client to iOS phones might be worth considering also because the user base of the iPhone is just as big if not even bigger than that of Android-based phones which would benefit the prediction quality but also make evaluation easier.

Bibliography

[1] Hoffmann A. Khan A.S. Building a case-based diet recommendation system without a knowledge engineer. *Artificial Intelligence in Medicine*, 27:155–179(25), February 2003.

[2] Luis Martínez, Rosa M. Rodriguez, and Macarena Espinilla. Reja: A georeferenced hybrid recommender system for restaurants. In *Web Intelligence/IAT Workshops'09*, pages 187–190, 2009.

[3] Moon-Hee Park, Han-Saem Park, and Sung-Bae Cho. Restaurant recommendation for group of people in mobile environments using probabilistic multi-criteria decision making. In *Proceedings of the 8th Asia-Pacific conference on Computer-Human Interaction*, APCHI '08, pages 114–122, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] 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*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

[5] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

Appendix A

Appendix

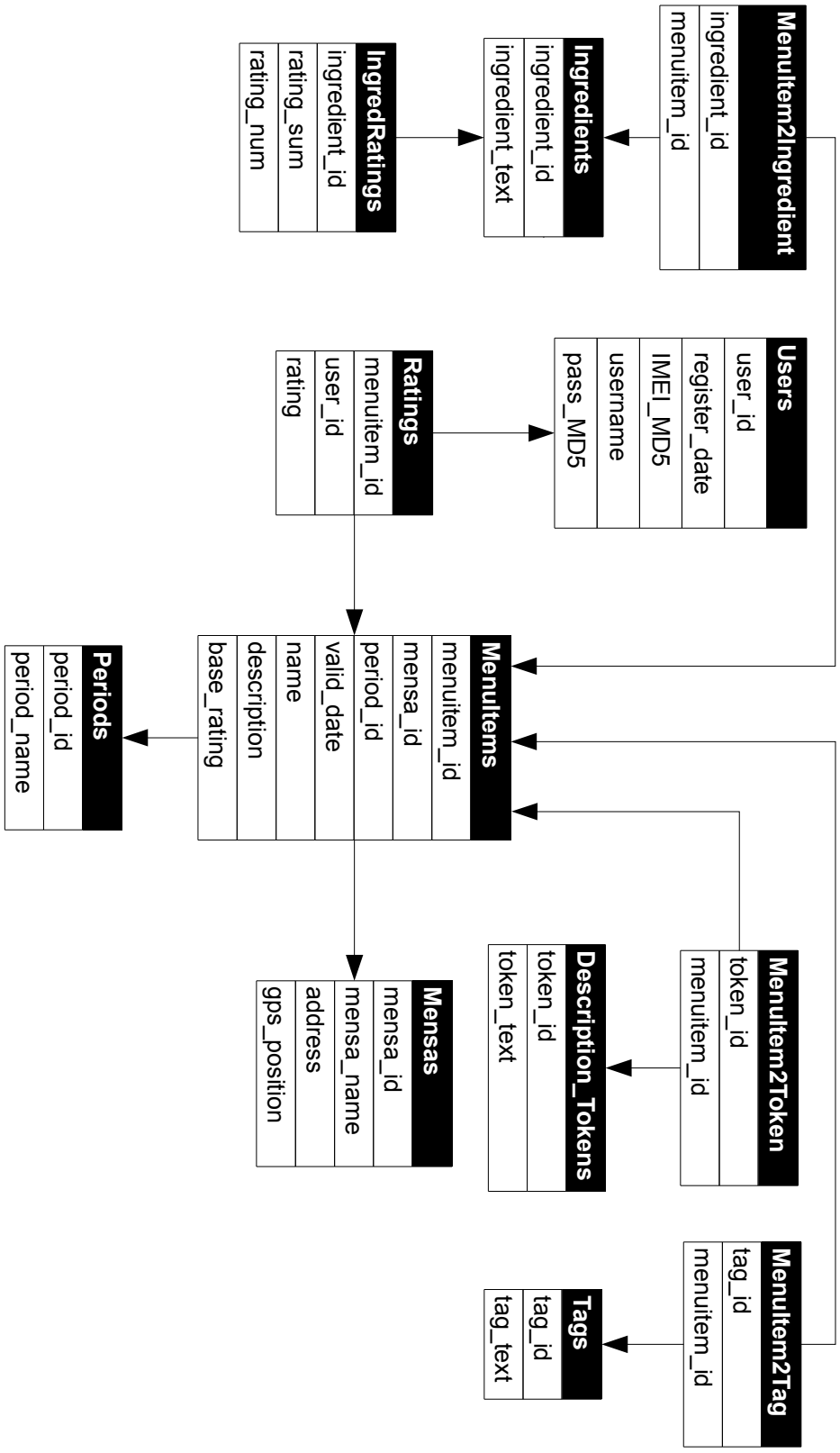


Figure A.1: Database structure