# Group Recommender System for Restaurant Lunches

ERIK HALLSTRÖM

KTH Datavetenskap
och kommunikation

Degree project in
Computer Science
Second cycle
Stockholm, Sweden 2013

KTH Computer Science
and Communication

# Group Recommender System for Restaurant Lunches

Master Thesis in Computer Science

Erik Hallström

*Supervisor at Ericsson:* Dr. Vincent A. Huang
*Supervisor at KTH:* Professor Anders Lansner
*Examiner at KTH:* Professor Jens Lagergren

August 15, 2013

## Abstract

A group recommender system for lunch restaurants is developed. The user interface is an Android application which is run on a smartphone. The system features a novel approach for implicit rating collection when a user browses a list of item descriptions. The individual recommendation is based on extracting and comparing tf-idf feature vectors of menu texts as well as individual rankings of the restaurants. The group recommender system works by aggregating the individual estimated scores of the members in the group, and in addition to this machine learning methods are used to capture the group dynamics in the group decision.

## Grupprekommenderingssystem för lunchrestauranger - Sammanfattning

Inom ramen för examensarbetet utvecklas ett system för grupprekommendation av lunchrestauranger. Användarapplikationen laddas ner och körs på en Android smartphone. Systemet använder sig av en ny metod för att samla implicit information från användarnas nyttjande av applikationen. Den individuella rekommendationen bygger på att extrahera och jämföra tf-idf feature-vektorer från menyernas text samt individuell rangordning av restaurangerna. Grupprekommendationen fås genom att aggregera de individuella estimerade betygen hos användarna i gruppen, och därtill används maskininlärningsmetoder för att simulera gruppdynamiken av gruppvalet.

**Acknowledgements**

I would like to express my sincere thanks to Dr. Vincent Huang at Ericsson and Professor Anders Lansner at KTH for helping and guiding me trough the course of this thesis.

# Contents

**I Introduction** **1**

**1 Purpose** **1**

**2 Discussion about decisions and choosing** **2**

**3 Item categorisation** **5**

**4 Background** **7**
- 4.1 Recommender systems . . . 7
  - 4.1.1 Collaborative . . . 8
  - 4.1.2 Content based . . . 14
- 4.2 Further extensions and machine learning techniques . . . 16
- 4.3 Group recommender systems . . . 17
- 4.4 Previous work . . . 18
  - 4.4.1 Individual recommendation . . . 18
  - 4.4.2 Group recommendation . . . 19
  - 4.4.3 Implicit ratings . . . 20

**II Method** **21**

**5 System setup** **21**
- 5.1 Android Application . . . 21
- 5.2 Server . . . 24
- 5.3 Database . . . 26
- 5.4 Parser . . . 28
- 5.5 Recommender system . . . 29

**6 Recommender algorithm** **29**
- 6.1 Discussion of choice of method . . . 29
- 6.2 Preference for restaurant . . . 32
- 6.3 Preference for food . . . 35
- 6.4 Global popularity . . . 43
- 6.5 Aggregating results . . . 43
- 6.6 Group recommendation . . . 44
  - 6.6.1 Group aggregation . . . 44
  - 6.6.2 Support vector machine for group recommendation . . . 44
  - 6.6.3 Aggregating group recommender results . . . 47

**III Results** **47**

**7 Performance** **48**

7.1 Ranking of browsed restaurants . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

7.2 Number of browsings in a browsing session . . . . . . . . . . . . . . . . . . . . . 52

7.3 Rankings of selected items and group browsings . . . . . . . . . . . . . . . . . 53

**8 User data** **55**

**9 Textual analysis of the menus** **58**

**IV Discussion and future work** **63**

**10 Discussion** **63**

**11 Future work** **65**

**V Conclusions** **66**

Part I

# Introduction

## 1 Purpose

The master thesis was carried out at Ericsson's Data Analytics Department. Their long time goal is to "analyse public and personal information, and use algorithms to make decisions for the user". The motivation for doing this came from personal assistants on mobile devices such as Google Now for Android and Siri for iOS. In particular they are interested in recommender systems, especially group recommender systems. My objective was to exemplify this by developing a recommender system for restaurant lunches on an Android application. The user interface, the Android application, was not developed by me, it was developed by students at Ericsson Labs in Bejing China. My task was to develop the backend of the system with the server, parser and recommender system. The employers full specification requirements for the front end and back end design of this system are explained in section 5 implementation. The problems raised from these requirements where the following:

**User information** No explicit user identifiers could be requested from the users, such as user names or telephone numbers. This meant that the group recommendation for a whole group had to be based on one person's input to the application. To solve this the employer suggested the *group browsing assumption* as outlined in section 6.2. The thought was that the system could simulate a group selection as a whole without knowing the preferences of an individual person in the group.

**Implicite preferences** The whole system was meant to require as little input as possible from the user. Similar to the user information no explicit user ratings could be collected for the items in the system.

**Individual recommendation** The first task was to provide an individual recommendation for a user based on earlier preferences.

**Group recommendation** After an individual recommendation was achieved a group recommender system was to be implemented.

To tackle these problems the following was done in this master thesis:

1. A discussion about in what context the recommender systems have been developed and how they achieve user happiness.
2. A literature study about individual recommender systems, group recommender systems and previous work in these fields.

3. A literature study about aggregating results and obtaining implicit selections from user behaviour.
4. Development of a server handling user data and public data.
5. Directing and specifying the application development of the front end Android application.
6. Developing novel method for implicit rating collection when a user is browsing a list of item descriptions.
7. Developing an algorithm for predicting individual ratings for the items.
8. Developing an algorithm for predicting group ratings for the items.

## 2 Discussion about decisions and choosing

On what grounds do we make decisions and choices? It all depends on what the choice is about and how big impact the decision will have on our lives. The tactics people use to tackle this problem will be very different depending on if the choice is about choosing a college, where to go on a vacation, what restaurant to visit or what music to listen to. Perhaps we do an extensive research on the subject, look up information and consider all the possible choices available. We may also ask a friend whom we trust or even a stranger for guidance and suggestions. Some people may take into account the opinions of more experienced people either by asking or reading reviews and articles written by them. Sometimes we just pick randomly or by feeling.

This process of decision making can be cumbersome and demand work. In order to choose a good restaurant for a special occasion the customer might look up all the restaurants in the neighbourhood, read the menus, weigh in own preferences as well as the preferences of the dining guests. In order to pick a good insurance plan one might hire a professional that does this work for you and selects a suitable plan tailored to your needs.

Lately the transition into the digital age has lead to a vast increase in the number of products that can instantly be consumed on a daily basis. The number of choices has exploded. For instance take the activity of listening to music. Some decades ago the consumer had to go to a music store to purchase a record. At some music stores the customer was allowed to try listen to songs, but to take out a LP record or CD from the shelf and put it in a player takes some time, and the shop owner would certainly not let the customer continue listening to songs indefinitely. Today the user easily gets on to the Internet trough computers or mobile devices and listens to music with streaming services such as Spotify and Grooveshark. These sites have thousands of song titles available, far more than the old music stores. The user can consume hundreds of titles in a short time;

start listening to a song, jump forward, decide whether to continue to listen to the song or forward to the next. The decision to listen to a particular song will not require analysis or consultations since the impact of making a bad decision and choosing a song that doesn't suits the user's preferences will just make the user to jump to the next song. However, listening to a song takes some time and the user might have to listen trough a number of songs not of the user's liking before a good song comes along.

The same phenomenon applies to movies, In the beginning of the 20th century there was only one, or a couple of films that were shown on a cinema a particular evening. Likewise in the early days of television there were only a handful of stations broadcasting, but that has ever since steadily increased. Later on video technology and video rental shops made the choices plentiful, on to the Internet age, where websites such as Netflix and Hulu have provided their customers with on demand movie streaming. As with most music streaming services the memberships are in the form of subscriptions, which means that the users don't have to worry about expenses along the way. Sometimes the movies or TV-shows are paid for by ads, meaning that the users first have to look trough an ad clip before being able to watch the content. However, the difference between music and movies is that the average song length is only a couple of minutes, where for a movie or TV-show the timespan is usually hours. It might take a while for the user to determine whether the currently watched movie is in the user's liking or not. Having the user to glimpse trough a number of movies, in a similar fashion as described above with music, will take time and perhaps require the user to look trough several ads. One could argue that the movie watcher could look at the trailers of the movies instead, but the approval of a trailer does not necessarily coincide with the approval of the actual movie and it also takes time to look trough trailers.

Another ground breaking revolution that came with the advent of the Internet and world wide web, was online shopping and e-commerce for the retail businesses. A customer can browse trough a large number of products from the websites of the vendors or use shopping search engines to find certain products or brands available. When the decision is made to purchase a certain item, the costumer pays for the product via an Internet bank, whereupon the product is shipped to the costumer's address. One difference between shopping products online and streaming music or movies is that the time to consumption a is couple of days for the shopping, where for streaming the consumption starts immediately after selection. The key thing for shopping online is that the costumer mainly base the decision of what item to buy by reading descriptions and looking at pictures of the products. It resembles the activity of watching movie trailers as described above; an approval of a picture or a description of a product might not resemble the approval of the actual product after a while of usage, even though they commonly are correlated. However, a user is sometimes able to try a certain number of products with an open purchase and return it with the money back if the customer is not satisfied. This procedure is rather exhausting and time consuming, and matches the activity of try watching movies or try listening to songs as described earlier. Also the consumption time for using the products can greatly differ depending on what item one buys. After a sip of wine one will immediately know if it is good or bad according to ones tastes, where for a car the life cycle is years.

The distribution of electronic books were made efficient by the Internet and allowed them to be introduced to a mass market. An electronic book is a book in digital form readable on devices such as computers, tablets, e-readers or mobile devices. Early on they where published on disk storage mediums like CDs or floppy disks. By then the only advantages of e-books from traditional books were that they were easy to carry and cheaper to produce, but the disadvantages where and are still several. By now books and magazines are published on the Internet and downloaded directly to special e-book readers. Online book retailers like Amazon both offers e-books and books in the traditional format. The difference from streaming services is that the digital content in the books is generally sold for each usage. The consumption time for a book is several hours, more than for the average film. Therefore the customers read reviews in order to get an idea of the content in the book. The e-book seller might also provide the consumer with a sample of the book, a couple of pages that can be read for free. But what really has changed is the time until consumption, the user can download the book in an instant and start reading, there is no need to go to the bookstore or library and start looking around on the shelves.

No matter if it is music, movies, e-books or other items, the common denominator is, as stated before, that the number of choices are huge with the Internet. Also the items that resembles the consumption; trailers, samples, reviews or descriptions are highly abundant and can easily be accessed. The time until consumption both for the content itself, as well as the reviews and samples has also vastly decreased. Before one had to pick up a paper or magazine that made a review of a product, or a catalogue that contained the item, look at the table of contents and scroll trough the pages, where today we simply input a search term in the search engine. With these many choices it is impossible for the user to consider them all, and one can easily be overwhelmed.

Since the subject of this master thesis is about restaurant recommendations, it is suitable to mention the activity of choosing a good restaurant when dining out. The world wide web has not really brought a great change to this particular pursuit. The things that are consumed in a restaurant is not only food, but also the service and dining environment, which all adds to the user experience and approval of the actual food. The consumption time of a meal can be from a couple of minutes to hours, and the time from decision until consumption is in the same order. There are actually two decisions being made; what restaurant to go to and what particular dish to order. The first of these decisions is usually made by a group of people, since we often tend to go out and eat together with others. The second is more of an individual choice. After a restaurant has been selected the party must transport them selves there. When the group reaches the restaurant they might get a feeling of whether they like the restaurant or not, they can look at the menu and decide whether to stay or to look for another. As before, this procedure is time consuming and adds to the time until a definite selection of a restaurant has been made. The selection of a dish is done by looking at descriptions or pictures of the actual items that are available to order in the menu. They resemble and describe the items just like trailers or reviews, and the time to consider the menu is far less than testing all the dishes. One could imagine other ways for the restaurant guests to form of an opinion of the dishes, they could for instance taste samples of the courses, but that would be a bit cumbersome and possible expensive for the restaurant owner.

## 3 Item categorisation

Given these examples, and in order to categorise the selection and consumption of different items I am suggesting these eight factors that are essential to understand the nature of the decision making. In spite of an extensive literature study I have not been able to find anything equivalent in the literature. To the best of my knowledge this classification is unique.

1. Cost of consumption
2. Time until consumption
3. Time of consumption
4. Number of choices
5. Impact of the decision
6. Influx of new items and item uniqueness
7. People involved
8. Consumption type

The *cost of consumption* is how large effort the user pays for using the product. Usually it is money, either a one time payment or a subscription, but it may also be a one kilometre walk down town to a nice restaurant, while going on an empty stomach.

The *time until consumption* is the time from the selection of an item has been made until it is consumed by a user. This one kilometre walk mentioned above both resembles an increase in cost of consumption and time until consumption. For the streaming services and online video browsing the time until consumption is zero, whereas for the online shopping a couple of days.

As noted earlier, different products take *different time to consume.* This fact has not changed after the Internet revolution. The time it takes to listen to a song is in the order of minutes, a movie several minutes up to hours. The time it takes to finish a meal at a restaurant is in the same timespan as with movies. Reading books can take hours and days, depending on the reader. A key thing here is, as we shall see later, the time it takes until a user has formed an opinion on the item being consumed, whether it is likeable or not.

The *number of choices* is simply the number of items a consumer can choose from. Note that this can be tens of thousands in case of music, but only a handful if it is dishes on a restaurant menu. The number of items available also changes with time, in case of music or movies almost all previous titles are still accessible, but for products in an online shop or restaurant dishes, this will be very dissimilar on different occasions.

Sometimes the *impact of the decision* may be very significant for the individual choosing between the options. Imagine a high school student choosing between universities or a CEO taking potential customers out for dinner. These types of decisions are very important and the chooser will certainly spend more effort in exploring the different possibilities.

Some restaurants may serve new dishes every day, that they never have had on their menu before. In that case, each dish is unique and can only be consumed on one occasion. For books on the other hand, the providers get in new written books from time to time, but each book can be read by a larger group of people. That will affect the way in which we can collect rating about the items, as we will see later. All these considerations are covered in *influx of new items and item uniqueness.*

The *people involved* resembles the number of people that are taking the decision together, their respective relationship and if there are other people affected by the decision. In the restaurant example mentioned above this factor is clearly of great importance. But it might also be group deciding to watch a movie together, or parents affected by their child's selection of collage.

The appreciation of a movie after watching it is much different from the opinion after watching the trailer or reading a review. That's why one has to make clear whether it is the description resembling the actual item that is being consumed or the item itself. The *time of consumption* and *time until consumption* is much different depending on if we are reading the menu describing the actual dish or eating it. These two measures are in the order of seconds for opening a restaurant menu and reading about a dish, but minutes for the actual dish being delivered, and hours for the consumption. Therefore these considerations are covered in the *Consumption type*.

# 4 Background

## 4.1 Recommender systems

A *recommender system* is a system that gives suggestions to a number of users what item they would like given either what they have liked in the past, what preferences they manually have specified in their profiles or other data that are available about the user [1, 2, 3, 4, 5]. The goal of a recommender system is to save time and increase satisfaction for the user. It is especially useful in the case of having the user to browse trough thousands of titles. This recommendation task is intelligent its nature, and could previously be made by a human that takes into account the personality and the preferences of the person choosing. In case of the *cost of consumption* and the *inpact of the decision* being high some would perhaps employ people for considering all the options and providing a recommendation. Otherwise the user might ask a friend that is well familiarised with all the items and the person choosing. On the contrary, with the Internet the providers are offering the consumer a much wider choice so that it is almost impossible for a human being to be acquainted with all the items.

Since this problem is algorithmic to a certain degree, it requires data to be collected and analysed, it would be desirable to have a program doing the suggestion. It would also be suitable to use it when the user browse items on the internet, i.e. recommending a a song, a move, a product in an online shop or an e-book, since the abundance of the items is so high and the consumer can easily get lost. The data of the user's preferences can either be collected explicitly, i.e. the user provides a rating right after consumption, or implicitly where the browsing behaviour is analysed and ratings are estimated automatically. [3, 6]

Regardless of if the activity is watching movies, listening to music or buying products online, one simple way of obtaining a recommendation for a particular song to listen, is picking from top charts. The position of the items in the top list can be based on sales or average ratings collected from polls. This type of recommendation could obviously also be done before the Internet age. In fact this is a simple form of recommender system, but the main problem with this approach is that it does not take in to account the fact that everybody's taste is different, and it can produce useless recommendations. Nevertheless, considering the popularity of an item one can greatly boost an existing recommendation, [7] a fact that is used in this recommender system.

It is now appropriate to introduce some notations, we have a set of $m$ users, $U = \{u_1, u_2, ..., u_m\}$ and a set of $n$ items that the users can consume $I = \{i_1, i_2, ..., i_n\}$. For these items a set of ratings has been provided from the users, $R = \{r_{ij}\}$. Here we have that $1 \leq i \leq n$ and $1 \leq j \leq m$. These ratings can be arranged in a $n \times m$ matrix $T$, but some the components will be empty since it is not certain that all the users have rated all items.

$$T = \begin{array}{c} \\ i_1 \\ \vdots \\ i_n \end{array} \begin{array}{c} \begin{array}{ccc} u_1, & \cdots & u_m \end{array} \\ \begin{pmatrix} \ddots & & \\ & r_{ij} & \\ & & \ddots \end{pmatrix} \end{array}$$

Generally the ratings are provided on a discrete scale by the user, $r_{ij} \in \{1, 2, 3, 4, 5\}$, and all these ratings are then stored in a database in the computers memory for later analysis. The task for a recommender system is now to estimate a rating of a product $i$ for a user $j$ where $\tilde{r}_{ji} \notin R$, i.e. $j$ has not rated the item before. To simplify the calculations, we denote the set of items that the user $a$ has already rated $Q(a)$, where $Q(a) \subseteq I$.

Since the first recommender systems emerged in the beginning of the '90s there have been two main approaches to calculate this estimate, these are *collaborative* and *content based* recommender systems [2, 4, 8, 9]. I will now talk about each of those and briefly describe how they work.

### 4.1.1 Collaborative

A collaborative recommender system only takes into account the user's previous ratings for the items, it does not attempt to analyse the actual items themselves. The goal is to find items that similar users have liked in the past. [4, 8, 9] If there are quite a few people that like to drink tea and eat scones, and if you seem to like tea, then a collaborative recommender system will recommend you to buy tea. It finds clusters of users that have similar tastes. The collaborative filtering can in turn be split into two subclasses, *memory based* or *model based* [3, 2].

**Memory based collaborative filtering** In memory based collaborative filtering the ratings database is held in memory and used directly for generating recommendations during runtime, there is no preprocessing. Two examples of memory based recommendation are *user based nearest neighbour recommendation* and *item based nearest neighbour recommendation* [3, 2].

**User based nearest neighbour recommendation** Here the goal is to find *similar users*, which have previously ranked the items similar to the each other. [10, 11, 12, 9] The task is to estimate a rating $\tilde{r}_{ia}$ for an item $i$ that a user $a$ has not rated before, and thus hopefully not previously consumed. We start by calculating the *average rating* a user $u$ has given to the rated products $Q(u)$.

$$\overline{r}_u = \frac{\sum\limits_{i \in Q(u)} r_{iu}}{|Q(u)|} \tag{1}$$

We now turn to trying to calculate the similarity, $sim(a, b)$, between a pair of users $a$ and $b$. There are several ways of doing this, one is the *Pearson correlation coefficient* which is shown in equation 2. [2, 4, 3, 9] We first have to find the set of items that has been rated both by $a$ and $b$, which is equal to the intersection between the two user's sets of rated items, $Q(a) \cap Q(b)$.

$$sim(a, b) = \frac{\sum\limits_{i \in Q(a) \cap Q(b)} (r_{ia} - \overline{r}_a)(r_{ib} - \overline{r}_b)}{\sqrt{\sum\limits_{i \in Q(a) \cap Q(b)} (r_{ia} - \overline{r}_a)^2} \sqrt{\sum\limits_{i \in Q(a) \cap Q(b)} (r_{ib} - \overline{r}_b)^2}} \tag{2}$$

The Pearson correlation coefficient will take values from $-1$ (very dissimilar users), to 1 (very similar users). As seen in equation 2 we take into account the fact that the user's rating behaviour is different. Some pessimistic people hardly ever gives a good grade to an item, where others give a five to every item they encounter. It is the relative rating of a product that we are interested in.

Next we will go about doing the estimation. The key concept is to select a threshold for the correlation coefficient, and thus select a subset of the users that are most similar to $a$. These are also called *nearest neighbours.* We select these and put them in a subset $N$, where $N \subseteq U$. The estimation can then be calculated:

$$\tilde{r}_{ia} = \overline{r}_a + \frac{\sum\limits_{b \in N} sim(a, b)(r_{ib} - \overline{r}_b)}{\sum\limits_{b \in N} sim(a, b)} \tag{3}$$

Equation 3 considers each neighbour user and calculates their relative approval of the item $i$. This approval is weighted by how close each neighbour is to the user $a$ we are doing the estimation for, and finally the expression is normalised.

**Item based nearest neighbour recommendation** The problem with the user based recommendation is that we have to consider a huge number of nearest neighbour users, which in large e-commerce sites can number up to millions. The system is thus very inefficient and can not be performed at runtime in a large scale application. The item based recommender system instead looks at the previous items users have rated and calculates the similarity between pairs of items. [11, 2, 3, 4, 9] Consider the items $c$ and $d$, where $c, d \in I$. The similarity between these items can be calculated as the scalar product between two row vectors in the ratings matrix $T$. Given these two vectors are denoted $r_{c,1:m}$ and $r_{d,1:m}$ we can write the similarity between the items $c$ and $d$ as following:

$$sim(c,d) = \frac{r_{c,1:m} \cdot r_{d,1:m}}{|r_{c,1:m}| \cdot |r_{d,1:m}|} \tag{4}$$

$$T = \begin{array}{c} \\ i_1 \\ \vdots \\ \\ \\ i_n \end{array} \begin{array}{c} \begin{array}{ccc} u_1, & \cdots & u_m \end{array} \\ \begin{pmatrix} \ddots & & \\ r_{c1} & \cdots & r_{cm} \\ & \vdots & \\ r_{d1} & \cdots & r_{dm} \\ & & \ddots \end{pmatrix} \end{array}$$

Note that the similarity between the items has nothing to do with what the item themselves contain, or what characterises them. It is just wether they have been *similarly rated* by the users.

What we calculate in equation 4 is the normalised scalar product of two vectors, and that is in its turn equal to the cosine of the angle between the vectors. Therefore this metric is labeled the *cosine similarity.* [2, 3, 4, 6] This metric is utilised within machine learning, information retrieval systems and recommender systems, and has shown to produce accurate results. Since there are only positive ratings in our example, this similarity measure only takes values between 0 and 1.

One problem with this approach is that it does not take into account that different users have different rating behaviour. This is, as previously described, accounted for when calculating the Pearson correlation equation coefficient in equation 2. Therefore we define the *adjusted cosine measure* as:

$$sim(c,d) = \frac{\sum\limits_{u \in U | c \in Q(u) \wedge d \in Q(u)} (r_{cu} - \overline{r}_u)(r_{du} - \overline{r}_u)}{\sqrt{\sum\limits_{u \in U | c \in Q(u) \wedge d \in Q(u)} (r_{cu} - \overline{r}_u)^2} \sqrt{\sum\limits_{u \in U | c \in Q(u) \wedge d \in Q(u)} (r_{du} - \overline{r}_u)^2}} \tag{5}$$

The sums in equation 5 are taken over the set of users which fulfil the relationship $u \in U \mid c \in Q(u) \wedge d \in Q(u)$. The user must both have rated the items $c$ and $d$ in order to contribute to the similarity. Actually this is also true for the traditional cosine similarity in equation 4, but in that case we simply leave out the terms in the scalar product for which a rating does not exist. As with the Pearson correlation coefficient in equation 2, this similarity takes values between $-1$ and 1.

We now attempt to predict a rating for an unrated item $c$ by a user $u$ using similarities between $i$ and the items $u$ previously rated $Q(u)$. The more similar one of these items are to the actual estimated item $i$, the more the $u$'s rating $r_{iu}$ would influence the end rating.

$$\tilde{r}_{cu} = \frac{\sum\limits_{i \in Q(u)} sim(c,i) \cdot r_{iu}}{\sum\limits_{i \in Q(u)} sim(c,i)} \tag{6}$$

Even though this item based approach is better suited for offline preprocessing than the user based recommendation [3], it still requires a lot of memory and computing power to perform the calculations, especially when there are millions of users and items.

**Model based collaborative filtering** The goal of model based collaborative filtering is to preprocess the data into a model that is used to get the recommendations during runtime. [2, 3, 4] If we in the previous example precomputed a (symmetric) matrix of pairwise item similarities for instant access, the system would turn to being model based. There are off course several ways of obtaining a model, I will now briefly describe two of these; *naive Bayes* collaborative recommendation and *SVD-based* collaborative recommendation.

**Naive Bayes** In our examples the ratings are graded on a 5-point discrete scale. Therefore we may treat the ratings as *classes* and use statistical methods to *classify* the rating of an unseen item. [2, 3]

Naive Bayes classifier is based on Bayes theorem, which states the relationship between probabilities of two events, $A$ and $B$ and their respective conditional probabilities.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{7}$$

Given that a rating is missing for an item $i$ by a user $u$, Bayes theorem is used to approximate the probability that the rating is equal to each of the scores in the scale. We then pick the score with the highest probability and assign it to $\tilde{r}_{iu}$.

$$\tilde{r}_{iu} = \arg\max_x P(r_{iu} = x | r_{qu} : q \in Q(u)) = \frac{P(r_{qu} : q \in Q(u) | r_{iu} = x) \cdot P(r_{iu} = x)}{P(r_{qu} : q \in Q(u))} \tag{8}$$

Now there are two things we can use, the first is that the denominator in equation 8 is independent of $x$. This means that we don't need to deal with it when calculating the probabilities, since we are only looking for the highest probability. The second is the assumption of naive Bayes: all probabilities are conditionally independent. This means that we take for granted that $P(r_{qu} : q \in Q(u) | r_{iu} = x) = \prod_{q \in Q(a)} P(r_{qu} | r_{iu} = x)$, which is indeed a naive assumption in reality, but naive Bayes has shown to give decent performance in the past. [2, 5]. With these two assumptions we conclude that

$$\tilde{r}_{iu} = \arg\max_x \left( \prod_{q \in Q(u)} P(r_{qu} | r_{iu} = x) \right) \cdot P(r_{iu} = x) \tag{9}$$

The next task is to figure out how to interpret these probabilities from the ratings matrix $T$. It is important to note that the item $i$ is unrated by $u$, so we have to use the ratings of other users $u'$ when we approximate the probability $P(r_{iu} = x)$ in equation 9. We simply count *the number of $x$ ratings given to item $i$ divided by the total number of ratings given to $i$.*

$$P(r_{iu} = x) = \frac{\sum\limits_{u' \in U : i \in Q(u') \wedge r_{iu'} = x} 1}{\sum\limits_{u' \in U : i \in Q(u')} 1} \tag{10}$$

Similarly when the expression $P(r_{qu} | r_{iu} = x)$ is calculated using only the *rows in the $T$-matrix where a user $u'$ has rated the score $x$ to the item $i$.* For each item $q$ and each user $u'$ we *count the number of $x$ ratings, divide by the total number of ratings to $q$ by $u'$* and assign the value to $P(r_{qu} | r_{iu} = x)$.

$$\prod_{q \in Q(u)} P(r_{qu} | r_{iu} = x) = \prod_{q \in Q(u)} \frac{\sum\limits_{u' \in U : i \in Q(u') \wedge r_{iu'} = x} \; \sum\limits_{q' \in Q(u') : r_{qu'} = x} 1}{\sum\limits_{u' \in U : i \in Q(u') \wedge r_{iu'} = x} 1} \tag{11}$$

By now all probabilities are explicitly defined, and we may calculate the most probable rating with equation 8.

**SVD-based recommendation** There are various types of dimensionality reduction techniques we can employ in order to reduce the amount of data we need to handle in the ratings matrix to better capture the model. [13] As previously stated, if the items and the users number up to millions, at the worst case we may have up to one trillion entries. Off course this is not true in reality, since every user only rates a small subset of the items. In SVD-factorisation the ratings matrix $T$ can be decomposed into the product of three matrices $U$, $\Sigma$ and $V$ as follows. [3]

$$T = U \; \Sigma \; V^T \tag{12}$$

Where $T$ is a $n \times m$ matrix, $U$ is a $n \times n$ orthogonal matrix, $\Sigma$ is a rectangular diagonal $n \times m$ matrix and $V^T$ is a $m \times m$ orthogonal matrix.

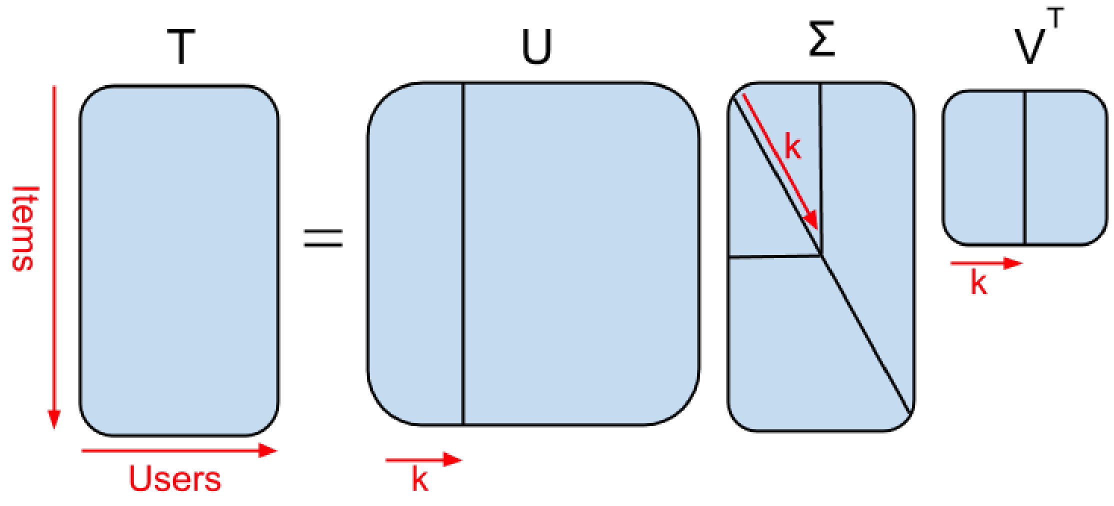


Figure 1: Shape of the matrices in SVD-decomposition and truncation

The $\Sigma$-matrix is a diagonal matrix, and only have values on its diagonal, while the rest is zero. These values are called *singular values* and it turns out that they are ordered in a decreasing manner, with the highest on the top-left corner. By choosing only the top-$k$ singular values and setting the rest to zero, and multiplying $U$, $\Sigma$ and $V^T$ back to a new matrix $T'$, we have created a rank $k$ approximation of the original ratings matrix $T$. This procedure is called *truncation*, and as shown in figure 1, we might as well discard all but the first $k$ columns of $U$ and $V^T$ since the others are equal to zero. We denote these new matrices with discarded columns as $U_k$, $\Sigma_k$ and $V_k^T$, which all only have $k$ columns.

It also appears that the rows of $U$ corresponds to feature vectors of items, and rows of $V^T$ corresponds to feature vectors of users. These can be put in a common user-item feature space, where similar users and items can be observed. Actually we only need to extract the $k$ first components from the row vectors of $U$ and $V^T$, thus only considering $V_k^T$ and $U_k$.

Now we turn to the actual goal of our recommender system; recommending an item to a user. There are many clever ways of doing this. Previously when we performed the factorisation of the ratings matrix $T$, we simply set all the missing ratings to zero. When wanting to obtain a recommendation for user $u$ we can look at all the feature vectors of items in the vicinity of $u$ in the user-item feature space. A straight forward measure for calculating this is the cosine similarity. We may then check the closest item for which no rating exist, and recommend it to the user.

If we get a new rating vector from a user, $r_{1:n,u}$, and want to put it on to the user-item feature space, we may project it using this formula.

$$\overline{r}_k = r_{1:n,u} \cdot U_k \cdot \Sigma_k^{-1} \tag{13}$$

Where $\Sigma_k$ is the matrix that results when only keeping the $k$ first singular values.

### 4.1.2 Content based

One disadvantage of a collaborative recommender system is that it never knows anything about the contents of the items that it recommends. A *content based* recommender system recommends and item to a user based on a description or the actual content of an item [2, 3, 4, 5, 8, 9]. In case of movies, the descriptions could be textual data such as genre, director, actors, play time and recommended age. For songs it could be artist, album, music style and release date. Obviously online news articles and books could have the same type of descriptions, but since the text contents of these items already are in the system, the whole text body could serve as the descriptor of the content. The music and movie files are also stored, so the visual and sound data of these could resemble them in a content based recommender system, but the problem with this is that sound and video take up a lot of space, several megabyte in case of music, and several hundred megabytes up to gigabytes for video. It would require an immense amount of processing power to do a recommender system based on this, which is why the vast majority of the content based recommender systems up to this date have been based around textual analysis.

**Term frequency - inverse document frequency** A standard way for characterising text documents in content based recommender systems is the *term frequency - inverse document frequency* measure [2, 3, 5]. It is based on the simple idea that if a keyword occurs frequently in a document, it better resembles the content of the document. But if the keyword in question frequently occurs in all documents in the document collection, the relative importance of the word is reduced.

Let us introduce some terminology, we have a set of documents $D$, which we name the *document corpus*, where $j \in D$ and there are $N$ documents in the corpus, $|D| = N$. In these documents we have a set of words, $I$ where $i \in I$. A frequency function, $freq(i, j)$, counts the number of keywords $i$ in a document $j$. The expression $\max\limits_{z} freq(z, j)$ will give the maximum frequency of a word in the document $j$. Next we have the function $n(i)$ which counts the number of documents in which the keyword $i$ appears. From this two measures are defined:

$$TF_{ij} = \frac{freq(i,j)}{\max\limits_{z} freq(z,j)} \tag{14}$$

$$IDF_i = log\frac{N}{n(i)} \tag{15}$$

Where equation 14 is the term frequency, and equation 15 is the inverse document frequency.[2] The term frequency varies from zero to one, whereas the inverse document frequency takes values between zero and $log(N)$. The term frequency inverse document frequency, or shortly TF-IDF for a keyword $i$ in a document $j$ are these two expressions multiplied together:

$$\text{TF-IDF}_{ij} = TF_{ij} \ \cdot IDF_i \tag{16}$$

Now we have calculated how well a certain keyword characterises a document. But our goal is to compare documents with each other, and that can be done if each document is described with a feature vector, $\overline{d}_j$. We construct this vector so that each component is a TF-IDF score from a term $i$, so there are in total $|I|$ components.

$$\overline{d}_j = (\text{TF-IDF}_{1j}, \text{TF-IDF}_{2j}, ..., \text{TF-IDF}_{|I|j}) \tag{17}$$

There are some preprocessing that can be performed on the corpus, which will improve the results. Two of these are *stemming* and choosing *stop words*. Stemming is modifying grammatically changed words in to their root forms and grouping synonyms together. For instance "potatoes" and "potato" will be grouped together as the same keyword. Choosing stop words is removing frequently occurring words such as prepositions, "and" and "me" that does not communicate anything about the content.

In order to obtain a predicted rating for an unrated document $d$, the cosine similarity can be used to find the $k$ most similar documents to $d$ for which a rating exists, and let them vote for the ranking on the current document.

**Roccio's algorithm** However, predicting the score of an unrated document is not the only way of recommending an item. Roccio's algorithm is a method of *relevance feedback* originated in information retrieval research in the '70s. It has also been used in recommender systems [5]. A user is shown a number of documents and specifies which of these are relevant and non-relevant. Each of these document's feature vectors are put in the sets $D_r$ and $D_{nr}$ respectively. A feature vector resembling the user, $\overline{q}$, is derived from these inputs.

$$\overline{q} = \alpha\overline{q}_0 + \beta\frac{1}{|D_r|}\sum_{\overline{d}\in D_r}\overline{d} - \gamma\frac{1}{|D_{nr}|}\sum_{\overline{d}\in D_{nr}}\overline{d} \tag{18}$$

Where $\overline{q}$ is the user tf-idf feature vector, $\overline{q}_0$ is the previous user tf-idf feature vector, $\alpha$ is the degree which the preference for the old documents decreases, $\beta$ is the amount of positive feedback and $\gamma$ is the amount of negative feedback. The algorithm updates the user vector $\overline{q}$ in a recursive fashion, where the procedure is repeated and the old user vector is set to $q_0$. Generally $\alpha+\beta+\gamma = 1$. The actual recommendation of an unread document is performed by calculating the cosine similarity between $\overline{q}$ and the unconsidered document's tf-idf feature vectors. The document that best coincides with the users interest that is covered in $\overline{q}$ is recommended to the user. An approach similar to Roccio's algorithm is used when recommending dishes to users in this recommender system.

## 4.2 Further extensions and machine learning techniques

The major disadvantage with collaborative recommender systems is that they require a lot of existing ratings in order to make accurate recommendations. This causes the systems to perform bad in the start, after being implemented, there are simply not enough similar users in these early stages. This issue is called the *cold start problem* [4, 3, 2].

A problem with the content based recommender system is that it only will recommend items within the same category. There is no diversification of the items recommended. After a while this can get tiering for the user, who perhaps wants to see something completely new.

There are several ways of coping with these drawbacks, one is to combine the systems, and these are called *Hybrid recommender systems*. [2, 3, 4] Research has shown them performing better than when only using one of the two standard approaches. In a way the prediction of a rating for an unseen item can be seen as a *machine learning* task. Algorithms such as Support vector machines, Artificial neural networks and Decision trees have been employed in the past, doing the rating regression with great success [5, 3].

## 4.3 Group recommender systems

A lot of activities, such as movie watching or dining, are often not carried out alone. A group recommender system aims to recommend an item to a group of people. It is often hard to measure the effectiveness of such a recommendation, since all group members will have different opinions about the recommended item. A great deal of psychology and philosophy can be incorporated into the problem, how to determine group satisfaction in a reliable way. [14] If for instance it is one of the group members birthday it is reasonable that he or she should not be disappointed with the selected item, and thus his or her rating should carry a greater weight. [15]

There are two main group recommending approaches that have been used in the past: group preference models [16, 17, 7, 15] and rating aggregation [10, 12, 1, 9, 18, 19, 20, 15]. This thesis uses both methods in the recommender algorithm. A group preference model uses previous selections and ratings of the users, builds a model and estimates one rating for each item to the group. Rating aggregation first estimates one rating for each user, and then aggregates these into a group rating. The way a group preference model is constructed is very problem specific. One could possibly merge keywords from the individual group members interests and find an item that matches most synonyms of these keywords. This has previously been done by M. S. Pera and Y.-K. Ng [7]. In this thesis the machine learning algorithm capturing group dynamics is building a group preference model. For the rank aggregation there are several approaches, a subset of these are *average*, *least misery*, *average disagreement* and *most pleasure*. These are all based on psychological evaluations, and are often aggregated into a final rating. In the formulas below $G$ denote the set of users that are doing the activity together, i.e. the group, and we have that $|G| > 1$.

**Average** This is the standard utilitarian approach, which maximises the happiness in the group. The average rating for an item is calculated. [10, 11, 12, 1, 16, 19, 20] One problem with this approach is that if one user has a large preference for an item and the other users dislike this item, it can still be favoured over an item where the ratings are much more even.

$$avr(G, i) = \frac{\sum\limits_{u \in G} r_{iu}}{|G|} \tag{19}$$

**Least misery** Least misery group recommendation minimises the individual unhappiness in the group. [10, 12, 16, 19, 15, 20] The group score of an item $i$ is set equal to the smallest estimated rating for the item among the users. One disadvantage with this is that if one item has a high rating for all group members except one, and another item has a much lower rating for all users except the one not liking the last item, which has only a slightly lower rating, the first item will still be selected. For example, the ratings for three users may in such case be $[10, 10, 3]$ for the first item and $[4, 4, 2]$ for the second item. Obviously it is better to choose the first item, since the last user seem to dislike pretty much both items.

$$lms(G, i) = \min_{u \in G} r_{iu} \tag{20}$$

**Average disagreement** It is fair for all the group members if an item is selected where most of the users have the same preference for the item. Minimising the disagreement about the items is a fair and diplomatic way of choosing. The average pair-wise disagreement, first introduced by S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu [10] can be calculated as follows:

$$dis(G, i) = \frac{2}{|G|(|G| - 1)} \sum_{u,v \in G, u \neq v} r_{iu} - r_{iv} \tag{21}$$

**Most pleasure** This measure is the opposite of least misery: it tries to maximise the individual happiness in the group. [12, 16] This method is not really a fair way of selecting, but can have some advantages if a user in the group is particularly important.

$$mps(G, i) = \max_{u \in G} r_{iu} \tag{22}$$

## 4.4 Previous work

This master thesis did consist of several subparts that needed to be investigated before an approach to solve the problem was selected.

### 4.4.1 Individual recommendation

The first recommender systems emerged in the early '90s, one of the first applications where to filter out interesting Usenet news messages to individuals. By then systems utilised collaborative approaches. Then recommender systems for music albums and videos where developed in the mid '90s. Because the usefulness of recommender systems, a great deal of the Internet businesses showed interest in the technology, and the pathway from research to commercialisation was fairly short. Following a dip during the Internet bubble in the early 2000s, the interest in the technology was then ignited by the 2006 announcement of the Netflix Prize, which was a competition for the best collaborative recommender systems that could predict user ratings for movies. [3] A data set was publicly available, and the best team was awarded a prize of one million US dollars. Often with online recommender systems, performance is a real issue, and a great deal of effort has been spent developing effective algorithms. Since this will not be a real issue in this thesis, the main focus was investigating the basics of how recommender systems work.

Two papers that summarised the previous work done in the field, as well as proposed possible future extensions where "Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions" by G. Adomavicius and A. Tuzhilin [2], and "Recommender systems" by P. Melville and V. Sindhwani [4]. In these articles cosine similarity, item based and model based collaborative filtering as well as the basics for content based recommendation and ti-idf weighting where explained. The way Bayesian classification can be used to predict ratings where also briefly outlined in [2]. "Content-based, collaborative recommendation" by M. Balabanovic and Y. Shoham gave a more in depth survey of the content based recommender algorithms, discussing item representation with tf-idf values, nearest neighbour methods, relevance feedback and Rocchio's algorithm, the learning of user models as well as content based naive Bayes. In "Matrix factorization techniques for recommender systems" by Y. Koren, R. Bell, and C. Volinsky [13] model based collaborative filtering was investigated, which was the basis of how the winner in the Netflix Prize solved the problem. "Content-based, collaborative recommendation" by M. Balabanovic and Y. Shoham also gave a brief introduction to collaborative and content based recommender systems and explained the implementation of a real recommender system which recommended web pages to Internet users. [8] A good book in the subject with several examples was "Recommender Systems: An Introduction" by D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. [3]

### 4.4.2 Group recommendation

The work that is most similar to this thesis is the "Pocket RestaurantFinder: A situated recommender system for groups" by J. F. McCarthy. It recommends restaurants to a group of people based on their predefined preferences. The system could be run on two different user interfaces, a kiosk or a handheld computer. Different from the system in this thesis, the distance, price and general cuisine of the restaurant was taken into account in the Pocket RestaurantFinder. However, the restaurants menu of the day where not considered and the group recommendation simply consisted of aggregating scores with only the average strategy.

A number of problem specific group recommending applications have been developed in the past. “State-of-the-art in group recommendation and new approaches for automatic identification of groups” by L. Boratto and S. Carta [11] and “Recommendation to groups” Jameson and B. Smyth [15] both gave a summary of the previous work, which were group recommender systems for music, books, web pages, travel and sightseeing destinations, TV programs and news items. Because the ad hoc nature of the problems, and that none of the previous works had enough similarities with this thesis, it was decided to only investigate a few of the mentioned systems further. The two articles also featured discussions about group recommender systems, their benefits and limitations. In addiction to this basic group score aggregation techniques such as average and least misery where explained.

Two examples of implemented systems were “Polylens: A recommender system for groups of users” by M. O’Connor, D. Cosley, J. A. Konstan, and J. Riedl [16], and “Tv program recommendation for multiple viewers based on user profile merging” by Z. Yu, X. Zhou, and J. G. Yanbin Hao [17]. The first of these recommended movies to a group by using a collaborative recommender system to obtain ratings and then aggregated these with the least misery measure. The second recommended TV programs to a group, and it implemented a model based group recommender system by merging user profiles. Another group recommender system which used a model based approach was “A group recommender for movies based on content similarity and popularity” by M. S. Pera and Y.-K. Ng. [7] It merged keywords of the interests of the individual users into a group profile. In addition to this a combination formula was used to boost the recommendation with the overall popularity of an item.

Group recommender systems are often used in online settings where millions of users and ratings are handled. That’s why efficiency is a real issue, and ”Group recommendation: Semantics and efficiency” by S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das [10], and C. Yu, and “Fast group recommendations by applying user clustering” by E. Ntoutsi, K. Stefanidis, and K. Norvag [12] examined this problem. Additionally, standard aggregation techniques were stated, and especially the average pair wise disagreement which was first proposed by [10]. Efficiency was not predicted to cause any complications in this thesis, so it was not taken into account.

### 4.4.3 Implicit ratings

The quest of collecting implicit ratings for an item is also very problem specific, but studies have been done in different contexts. In the article “Implicit interest indicators” by M.

Claypool, P. Le, M. Wased, and D. Brown [21] implicit ratings for web pages were collected, based on scrollbar, keyboard and mouse activities on the page. Also the total time spent on a page was recorded. The users were asked to provide explicit ratings in order to measure the performance. It showed that the time spent on the page was clearly correlated with the explicit rating. In "An adaptive personalized recommender based on web- browsing behaviour learning" by K. Takano and K. F. Li [6] another recommender system for web pages was developed by analysing browsing behaviour and collecting implicit ratings. The web pages browsed, terms copied onto the clipboard, pages bookmarked, pages printed and search keywords input to search engines etc. were recorded and used as input to the recommender system. Since there were few similarities between the previous works and this thesis, a completely new method needed to be developed.

# Part II
# Method

## 5 System setup

The goal for this thesis was to set up a system where a user could see the menus for the nearby restaurants in an Android app. The user was then recommended a restaurant based on preferences collected earlier. The system consisted of a client that ran on a smartphone device, a server, a database, a parser collecting data from a public website containing the menus and a recommender system. The overall structure is shown figure 2. I developed the backend of the system, all parts in the figure except the client.

### 5.1 Android Application

The application was developed at Ericsson labs in Beijing by students at Beihang university. It featured a list of the restaurants in Kista municipality, a suburb to Stockholm where Ericsson's headquarter is situated, see figure 3. When pointing at one of the restaurants, its menu for the day was showed to the operator. There was no direct recommendation of a restaurant for the day, the recommendation was more like a ranking system where the list of restaurants was ordered according to the users preferences.

There were two functionalities in the app, the first was the ability to obtain a group recommendation for a group of people eating lunch together by adding contacts from the contact list, figure 5. After the lunching friends were added, the ordering of the restaurants in the list was suppose to change accordingly. The second functionality was the ability to tell one of the lunch partners, supposedly not being present in the party, where the group went for lunch, figure 6. In that way the decision of the group was implicitly told to the

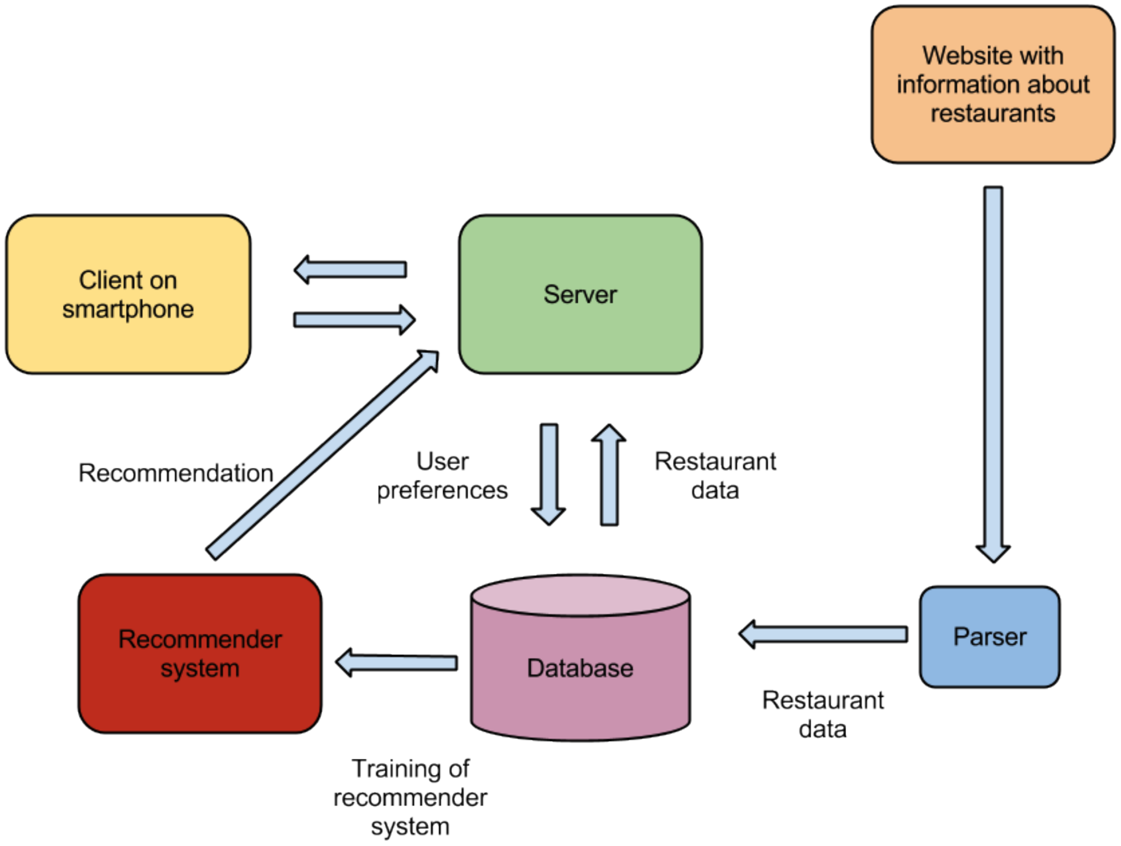
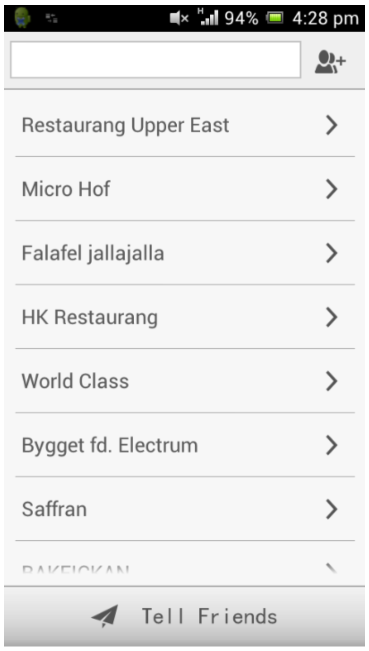


Figure 2: System setup for the recommender experiment

server, and could be used in the recommender system. Screenshots from the application are shown in figures 3 to 8.

The intended users of this application where perhaps colleagues eating lunch together, people that already knew each other. At the data analytics department in Ericsson several of the coworkers were having lunch together every day in one of the nearby restaurants, and there was always a discussion of what restaurant to visit. The aim for this app was that one of these people in the group would be the operator, he or she would then add the coworkers from the contact list and start browsing restaurants. The assumption was that the nearby coworkers would also look at the mobile device running the app, and influence the browsing behaviour of the operator. This procedure will be explained more in detail in the later section covering the recommendation algorithm. It often happens that one or several of the coworkers are stuck in meetings or other appointments, so the operator can tell that person where the group is going, as mentioned above.



Restaurang Upper East

Micro Hof

Falafel jallajalla

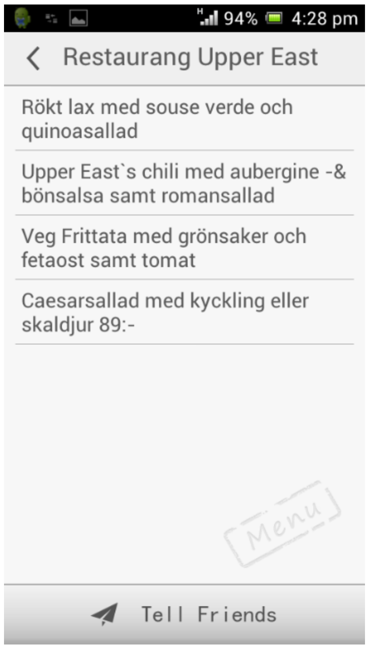
HK Restaurang

World Class

Bygget fd. Electrum

Saffran

Figure 3: Screenshot from the start screen of the app, a restaurant list. The recommender system ordered the list based on the user's preferences.



Restaurang Upper East

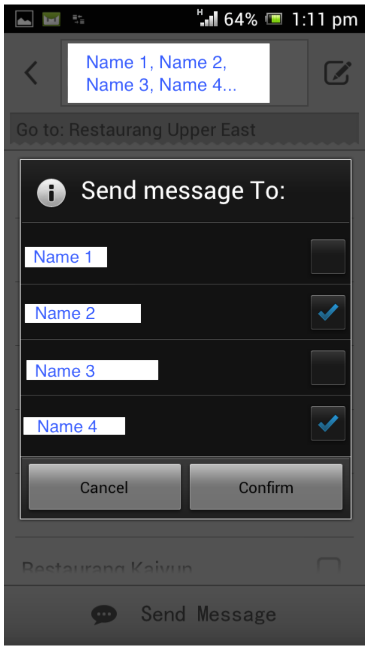
Rökt lax med souse verde och quinoasallad

Upper East`s chili med aubergine -& bönsalsa samt romansallad

Veg Frittata med grönsaker och fetaost samt tomat

Caesarsallad med kyckling eller skaldjur 89:-

Figure 4: Dishes on the menu for a restaurant. After the user pointed at one of the restaurants in figure 3, a list of the available dishes in the restaurant that day was provided to the user.



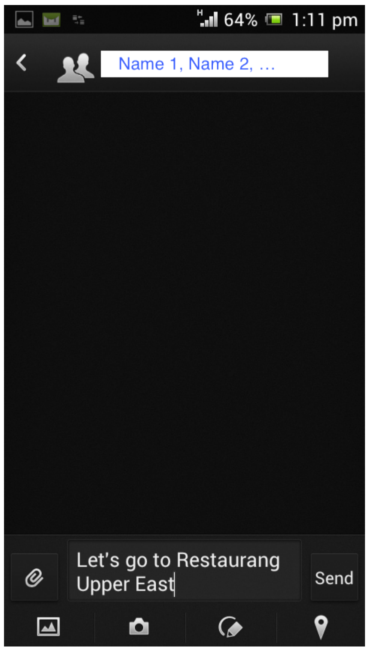
Name 1, Name 2, Name 3, Name 4...

Go to: Restaurang Upper East

Send message To:

| | |
|---|---|
| Name 1 | ☐ |
| Name 2 | ✓ |
| Name 3 | ☐ |
| Name 4 | ✓ |

Figure 7: A number of people from the list of lunching partners could be notified about the selected restaurant



Name 1, Name 2, ...

Let's go to Restaurang Upper East

Figure 8: The notification is sent by sms to the selected contacts.

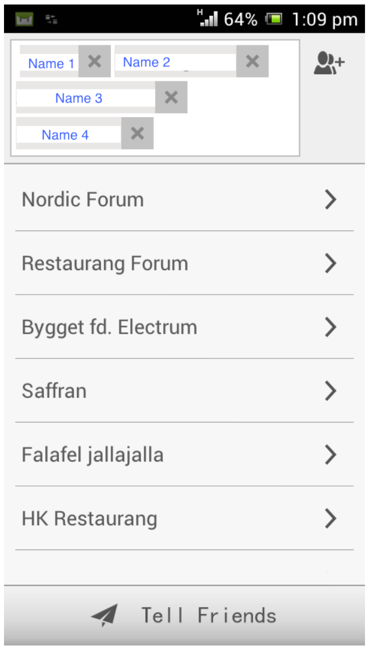


Figure 5: Friends that will eat lunch together are added from the contact book. If there is any information about the preferences of the friends, the ranking of the restaurants will change.

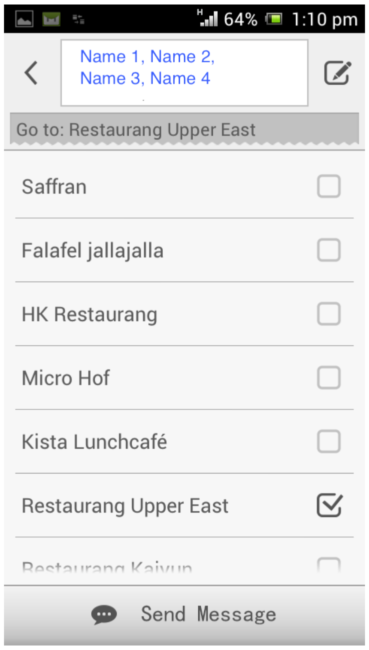


Figure 6: After a restaurant has been chosen by the party, the operator of the app may choose to tell one or several of the contacts in the list where the group is going.

## 5.2 Server

The server is responsible for communicating with the client and the database, and handling identification information from the users, such as telephone number, IMEI number[1] and email. There are three basic functions in the server, these are *getRestaurants*, *getDishes* and *writeSelection*. The functionality of the server is shown in figure 9.

**getRestaurants** The function requires a specified list of lunch partners as well as the operator that is using the app on the smartphone. The server calls the recommender program with the specified users and restaurants, and the recommender algorithm will then order the restaurants according to the user's preferences, and return the ordered list. The components of this function are shown in red in figure 9.

**getDishes** This function not only requires the list of lunch partners and the operator id, but also the specified restaurants. This function is called when the user points at a

[1]An IMEI number is a unique hardware identifier for a mobile phone, see section for further details.

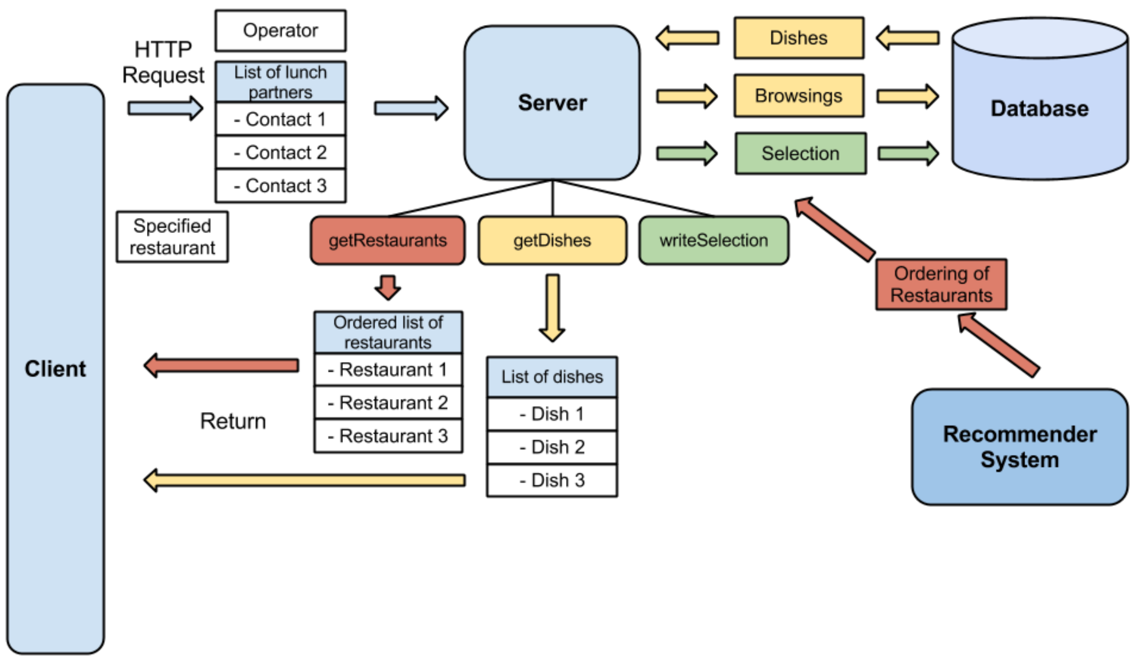


Figure 9: Basic functionality of the server

restaurant in the app. The server will read the dishes of the day from the database, which previously have been stored there in the morning. The function then returns the list of dishes (i.e. the menu) for the named restaurant and also writes an entry of this browsing occasion to the database. In that way the interests of the users can be tracked. The functionalities of this function are shown in yellow in figure 9.

**writeSelection** The function is called when the party has made up a decision of what restaurant to visit and wants to use the "Tell friends" function, as described earlier, to tell one person not present in the group. The function requires information about the group as well as the restaurant selected. The server writes the selection to the database, and this information is used in the algorithm to further boost the recommendation. The components of this function are shown in green in figure 9.

All the communications with the server are done trough HTTP-requests, and only plain text messages are returned. The Server is written as a Java Web Application running in a

Apache Tomcat Servlet Container. The access from the server to the database is obtained using the Java DataBase Connectivity driver, JDBC.

## 5.3 Database

There is a MySQL database management system running on the server, with a database containing a number of tables which the server writes to and reads from. The tables are named *id_consumption_table*, *browsing_history*, *restaurant_dishes*, *selection* and *estimated_selection*. In addition to these tables there are a few others that are used for debugging and data analytics purposes.

**id_consumption_table** There are several ways of identifying a user in the system. It was decided to use three of those, that are common for identification usage in smartphones. The identifiers selected were IMEI-number, telephone number and email. The IMEI, short for International Mobile Equipment Identity, is a unique hardware identifier for mobile phones. It is a 15-digit number that is tied to the mobile phone, and thus can not be changed, even if the mobile changes owner. If given access, the IMEI-number can be read by a application on the smartphone. If the user have specified his or her telephone number or email in the Google profile, this information can be read by the Android application as well. The application can also access information about the added lunch partners, their telephone number or email if they are specified in the contact book.

All this information is put together in the id_consumption_table. Whenever a user identifier enters the system, the server searches trough the table and sees whether the identifier has been registered before. If this is true, the user will be assigned the corresponding internal id number as shown in table 1. If the search didn't return anything, a new entry is created in the table for the user with the identifiers specified.

The reason for using these many identifiers is that in order to do a good recommendation for a group of people, we need to know about the preferences of as many as possible in the group. Since there is no explicit ratings collected, the only thing we can study is the browsing history. When a user browses the restaurants the IMEI-number will most certainly be collected, and hopefully have either the telephone number or the email been specified in the Google profile. If this same person is later added as a contact with another person operating the application on another device, in the best case one of the telephone numbers from the contact book is matched with the telephone number specified in the Google profile. We can then load the browsing history of the user. If this event occurs it is called a *contact-operator match*. Obviously IMEI-numbers can not be read from the contact book.

There can be several telephone numbers in the contact book, so a number of columns are allocated to save those for a user. Only one email and one IMEI-number is allowed for identification of a user. The structure of the table and examples of the entries are shown in table 1.

Table 1: Example of an id_consumption_table. There are in total 10 identifiers where the first has been assigned to the IMEI-number, the second has been assigned to email address, and columns three to ten are saving telephone numbers. A contact-operator match has happened in the first row, since there are several telephone numbers tied to a IMEI-number. In the second row an operator has not specified the telephone number or the email, so we can't continue from there. In the third row a contact has been added with only one specified telephone number, that has not yet been matched.

| Internal ID | Identifier 1 | Identifier 2 | Identifier 3 | Identifier 4 | ... |
|---|---|---|---|---|---|
| 1 | 358506046839050 | me@example.com | +46733443222 | +4623423423 | ... |
| 2 | 142342352324234 | NULL | NULL | NULL | ... |
| 3 | NULL | NULL | +46790977096 | NULL | ... |

**browsing_history** This table is written to as soon as a user requests to see the menu from a restaurant. While the server returns today's menu for a specified restaurant, the internal id number for the operator and added contacts, the restaurant name and a timestamp for the browsing are all stored in the table.

In addition to this, when the menu from a named restaurant is requested from the server, the server sends an internal request for the ranking of the restaurants, using the user and lunch partner identifiers as parameters. In that way the server figures out what position the actual restaurant had in the list. This is used for boosting and evaluating the performance of the recommender algorithm, as we shall see later. An example of how the table would look like is shown in table 2

**restaurant_dishes** This table is used to store all the dishes for each date and restaurant in the database. There are only three columns; date, restaurant name and dishes. For more information about this table see the next section about the parser.

**selection and estimated_selection** These two tables have exactly the same format as the browsing_history table in figure 2. The selection table is written to after the "Tell

Table 2: Example of how browsing_history table would look like. The timestamps are in the format *day.month.year.hour.minute.second*. The ranking of the browsed restaurant is stored in the last column. The operator and people in the lunch partner list are all specified with internal IDs.

| User | Contacts | Timestamp | Restaurant | Ranking of selected restaurant |
|---|---|---|---|---|
| 1 |  | 13.6.2013.10.33.43 | Restaurang Upper East | 3 |
| 1 | 2,3 | 13.6.2013.10.34.2 | World Class | 2 |
| 3 | 4 | 14.6.2013.11.10.19 | Secret Recipe | 10 |

Table 3: Example of how restaurant_dishes would look like. The date is in the format *day.month.year*. The different dishes are separated with three pound signs.

| Date | Restaurant name | Dishes |
|---|---|---|
| 18.6.2013 | Scandic Victoria Tower | Raggmunk med stekt fläsk och rårörda lingon ### Rimmad lax med dillstuvad potatis ### Varm bulgursallad med fetaost, aubergine och citrondressing 135:- |
| 18.6.2013 | Upper East | Ört -& citronhoki med fänkål och kokt potatis ### Kalvfärsbiff med chevré och grynsallad ### Veg Indisk lins -& okragryta med korianderkräm ### Caesarsallad med kyckling eller skaldjur 89:- |

friends" button has been pressed, and the user have specified a restaurant to go to. Internal ID numbers for the individuals in the group that are eating together are saved to the table, as well as the name of the restaurant and a timestamp. The ranking the selected restaurant had in the list is also relevant, so it is saved as well.

If there is no information about where the group went, a restaurant selection has to be guessed. This is done in the recommender algorithm, which I will describe later. This simulated selection is saved to the estimated_selection table.

## 5.4 Parser

The information about the menus of the restaurants are publicly available on a website. When a menu from a restaurant is requested from the server, the information is not collected from this website, it would take too long time, since the HTML-code has to be parsed at runtime. Instead information about the menus are stored each morning in the database, namely in the restaurant_dishes table. The server reads directly from this table when providing the menus.

The parser is responsible for downloading and storing the menus in the database. The program is written in Java and utilises the Jericho HTML Parser library. It connects to the database with the JDBC framework. The process is performed each morning with the cron-job scheduler which executes the .jar file. The HTML-parsing of the dishes are sometimes not very accurate, since there is not a standard way of separating the different dishes for a given restaurant on the website. Sometimes that is only done with a <BR>-tag. But the different restaurants and their respective menus are well structured in tables on the site, so they can easily be extracted.

## 5.5 Recommender system

The ordering of the restaurants in the application is handled by the recommender system. It is written in Python and utilises the Scikit-learn machine learning library [25]. There are two components in this system, the Train and save and the Recommender. The overall structure of the program is shown in figure 10.

**Train and save** Every morning after the menus have been downloaded for the day to the restaurant_dishes table, the preferences for each user and restaurant are estimated by the recommender system and saved in a data structure. The program then trains a Support Vector Machine, which aids the group recommendation. These two objects are saved with pickle, which is the standard mechanism of object serialisation in Python. Train and save is put as a cron-job and is performed in the morning just after the parsing has finished.

**Recommender** The recommender program is run by the server when it receives a call to the getRestaurants function. The server specifies the people in the group and the recommender returns a list of ordered restaurants. Both the Support vector machine and the preference objects are deserialized from file and used by the program at runtime.

# 6 Recommender algorithm

## 6.1 Discussion of choice of method

The previous work that has been made in recommender systems, as explained in the previous chapter, has mainly circled around predicting a score for a not yet seen item. The properties that characterise these systems is that the *number of choices* is huge and the

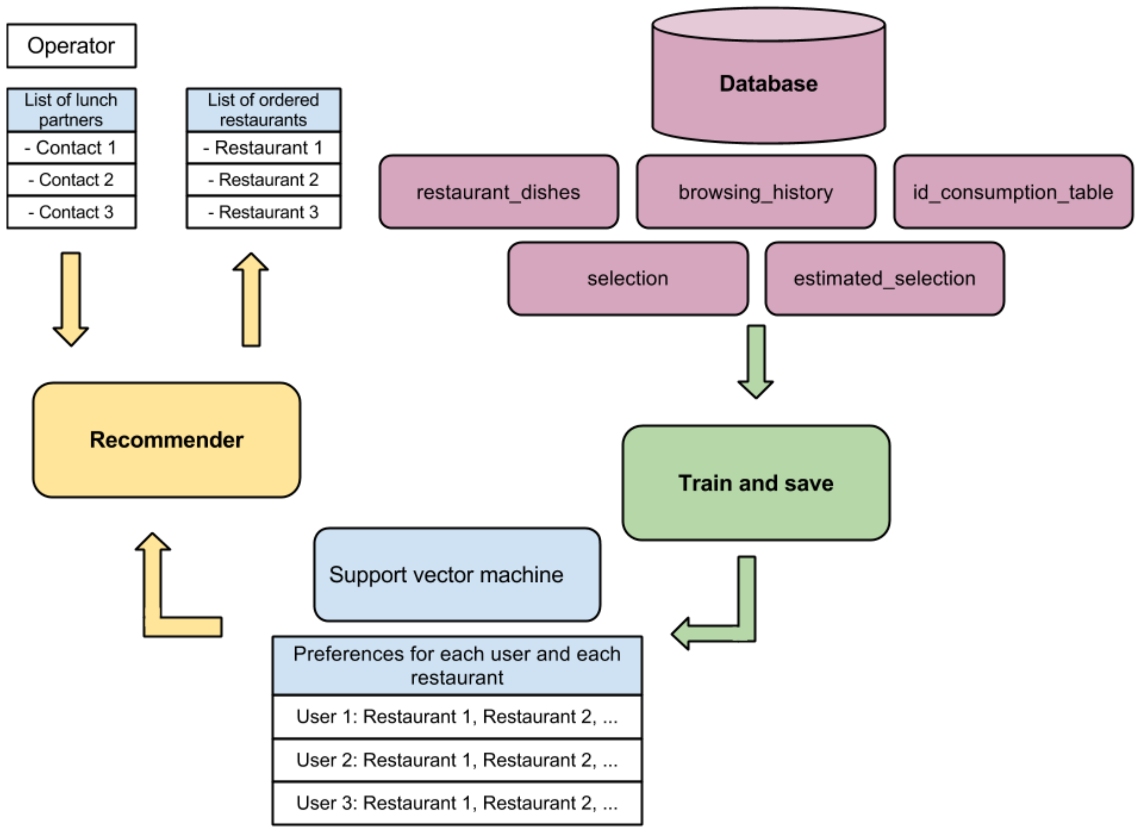


Figure 10: Functionality of the recommender system

*consumption type* is such that the actual item, not a description is consumed. In most cases an explicit rating is provided to the system right after the actual item has been consumed, or at least after a while when the user has gotten an opinion about the item. Both in music and movie recommender systems, which are the two most prominent areas the technology is used, is there is a relative low *influx of new items.* The titles are mostly available indefinitely, and explored by many users. For these two mentioned services *impact of the decision* and the *cost of consumption* is low. Beginning to watch a bad movie or reading a uninteresting news article just wastes a little bit of time for the user.

This restaurant recommender system has several major differences from the other systems mentioned. The most important is perhaps that the ratings are collected implicitly from the browsing behaviour when looking at the *descriptions of the items*, before the user has consumed any items at all. Another characteristic is that a user might have preferences for either a dish or a particular restaurant, or both. The *number of choices* are low, there is even a possibility that the user can explore all the descriptions of the items, i.e. browse trough all the restaurants. For many of the restaurants there are constantly new dishes arriving on the menu, so there is a high *influx of new items*. The *impact of the decision* can be rather high, eating good is important for the work efficiency, so the user may think about the restaurant selection twice, rather than just look at the restaurant's ranking on a mobile app. The items, i.e. the restaurants and the same dishes are supposed to be consumed several times, where movie recommender systems only recommends unseen items. However, there are some similarities with this restaurant recommender system and the previously mentioned systems. In this setting, the *cost of consumption* and *time of consumption* are very low for reading the menus, whereas the *time until consumption* is instant.

Because of the ad hoc nature of this problem, a completely new approach had to be developed. After all, the methods that are presented earlier, especially in the group recommender systems, are all based on psychological arguments.

There are two expected benefits of this system, the first is to save time for the users, to provide examples of relevant restaurants and menus for the group to discuss, and in a relative short time come up with a decision. The second benefit is to notify users about restaurants and especially dishes they have good preferences for, and didn't know was available that particular day. Given these premises, the standard collaborative algorithms, as explained in the earlier section, are not suitable. They will suffer the cold start problem, since the time scale of this experiment is too short. Besides, looking at the browsings the system will relatively fast get an opinion about each user's preference for each restaurant , since there are so few. It is also appropriate to analyse the menu text and get the liking of a particular dish. Finally all these ratings have to be aggregated in a smart way, into a single rating for an individual or a group. In addition to this, it would be good to have a model capturing the group selection dynamics.

**Definitions** Before proceeding to the algorithms, let's introduce some terminology. We assume that a user uses the application only once a day before having lunch. This is called the browsing session, and the number of browsed restaurants is denoted $i_{max}$. By looking at all the browsing sessions at different days one can calculate the mean number of browsed restaurants for a session, and this denoted $i_{mean}$. The order a particular restaurant had in a browsing session, in its most recent browsing is denoted $i$. Note that if a restaurant is browsed twice, $i_{max}$ is not increased and $i$ is simply updated to the restaurants last position in the browsing session.

Similar to the restaurant browsing order defined above, a variable $j$ is introduced as the actual browsing order. With the restaurant browsing order several browsing occasions on a restaurant will count as one. If a user checks out restaurant A, B and C in sequence one time each, and then looks at restaurant A several times, $i_{max}$ will still be three and the browsing order for restaurant A will become $i = 3$. This variable $i$ is tied to a particular restaurant in a browsing session, where the variable $j$ is tied to a browsing (pointing at a a restaurant and looking at the menu) in a browsing session. Similarly $j_{max}$ is the total number of browsings in the session. In the example $j_{max} = 5$ and $j$ is 1,4 and 5 for each of the browsings of restaurant A.

The time and date when a browsing or selection of a restaurant took place should influence the rating. The time of the user's first usage of the app is $t_0$, the time of the current date is denoted $t_{now}$, and the time when the browsing occurred is $t_{browse}$.

The ranking a particular restaurant had in the list when the user browsed or selected it is denoted $n$, and the total number of restaurants available that date is $n_{tot}$.

The time from a user has requested the menu from a restaurant until a new menu is requested is extracted from the browsing table in the database. This number is then divided by the text length (number of characters) in the browsed menu to get the normalised browsing time $t_{norm}$. The mean normalised browsing time for a user is calculated over all browsing occasions and is denoted $t_{mean}$. There is a threshold of 60 seconds, if the browsing time is larger than that the time is not extracted from the database. This is the case for the last browsed restaurant, since we can only measure the time between requests.

## 6.2 Preference for restaurant

In order to calculate the implicit rating for a restaurant by a user, data is collected from the browsing_history and selection table. There are three things accounted for: order of the browsing, time since browsing occurred, and ranking of browsed restaurant.

**Order of browsing** It is reasonable to think that the ordering of the browsings reflects something about the users preference. If a user always begins looking at a particular restaurant, it is perhaps more interesting than the last one examined. The earlier the user checks a restaurant in a browsing occasion the more importance it deserves. The weight capturing this, $w_1$, is defined as

$$w_1 = \frac{i_{max} - i}{i_{max}} \tag{23}$$

where $i$ and $i_{max}$ are previously defined and $w_1 \in [0, 1]$

**Time since browsing or selection** The further back in time a browsing occurred, the lower importance it should have when calculating the restaurant preference. The opinions are assumed to change over time, and the most recent selections should have greater relevance. The weight $w_2$ is defined as

$$w_2 = \frac{t_{browse} - t_0}{t_{now} - t_0}, \quad t_{now} \neq t_0 \tag{24}$$

where $w_2 \in [0, 1]$. The differences in time can be translated to any unit, preferably seconds, since it is a fraction that is being calculated. It is guaranteed that $t_{now} \neq t_0$, since calculation of ratings takes place in the morning before new data is let into the system.

**Ranking of browsed restaurant** If a user scrolls further down to the bottom of the list, just to see a particular restaurant, it should mean that it is especially interesting to the user. The top parts of a scrollable page gets much more attention, and are much more prone to selection. Define the weight $w_3$ as

$$w_3 = \frac{n}{n_{tot}} \tag{25}$$

Where $n$ and $n_{tot}$ are defined earlier and $w_3 \in [0, 1]$

**Group interaction** If the number of people in the group increases, there is a higher probability that one of several of them had to compromise when selecting a restaurant. To capture this we define an interaction weight:

$$w_{interaction} = \frac{1}{|G|} \tag{26}$$

Where $|G|$ is the number of people in the group (the number of people in the lunch partner list, with the operator included) and $w_{interaction} \in [0, 1]$.

**Calculating scores** The sources for calculating scores for the restaurants are taken from the selection and browsing_of_restaurant tables in the database. If the set of selections by all users in the database is denoted $S$, where $s \in S$ and the set of restaurant browsings by all users is denoted $B$, where $b \in B$, we can define the score $r$ for a user $u$ and a restaurant $i$ as

$$\begin{aligned}\tilde{r}_{iu} = &\sum_{b\in B:restaurant(b)=i\wedge operator(b)=u} \left(\frac{1}{2}+\frac{1}{2}w_{interaction}(b)\right)\frac{w_1(b)+w_2(b)+w_3(b)}{3} + \\ &\sum_{b\in B:restaurant(b)=i\wedge u\in friends(b)} \frac{w_{interaction}(b)}{10}\frac{w_1(b)+w_2(b)+w_3(b)}{3} + \\ &\sum_{s\in S:restaurant(s)=i\wedge operator(s)=u} 10w_{interaction}(s)\frac{w_2(s)+w_3(s)}{2} + \\ &\sum_{s\in S:restaurant(s)=i\wedge u\in friends(s)} 10w_{interaction}(s)\frac{w_2(s)+w_3(s)}{2} \end{aligned} \tag{27}$$

where $restaurant(x)$, $user(x)$ and $friends(x)$ returns the restaurant, user or the set of people in the lunch partner list for a specific browsing or selection $x$. The order of browsing measure, $w_1$, will not give any further information for the selection, there is normally only one selection for a group on a particular day. However, the ranking a selected restaurant had in the list, and the time since it was selected should affect the preferences for the selected restaurant. It is important to mention that the weights in the above formula are all dependent on the data from the actual browsing, selection, users participating and the browsing session. The motivation for this formula is explained by the group browsing assumption outlined below.

The unnormalised value that is derived in equation 27 is then normalised by dividing each rating with the norm of the total ratings vector $\overline{r}_u$ for a user, in which each component is a rating for a restaurant. The top-script in the equation below refers to index, not exponent.

$$r_{iu}^1 = \frac{\tilde{r}_{iu}}{|\overline{r}_u|} \tag{28}$$

**Group browsing assumption** It is assumed that there are three main usage forms of this application. The first is an individual user browsing the restaurants with no people added in the lunch partner list, the second is an operator and a group of lunch partners looking at the same device while browsing and the third is a single operator with added lunch partners in the list, browsing alone. In the later two cases, the operator should still have the largest influence of the browsing behaviour, but if there are several lunch partners looking at the device, or if the operator is planning a lunch with others, their opinions will affect the operators decision. If for instance, the operator knows that person A really likes restaurant B, the operator might check it out. The more people there are in the lunch partner list, the more each one in the group have to compromise. All this is covered in the interaction factors in equation 27. We always want to increase the weight for a restaurant if an operator browsed it, regardless how many people there are in the group, but the added score should still decrease with the number of people. It is why one half is added before the addition of one half times the interaction weigh in the first term in equation 27.

It is reasonable to think that one of the lunch partners will only have a small influence of the restaurant browsing, the operator is after all holding the device. Therefore the second term in equation 27 is divided by ten, if we have information about the browsings when the lunch partner was the operator, these should count much more when calculating preferences for restaurants.

It is furthermore assumed that all the people in the group roughly have the same influence on the restaurant selection, so if such a selection exist, the same score is added to the particular restaurant for both the operator and the lunch partners. These scores are multiplied by ten, as can bee seen in the third and fourth term in equation 27. A selection is telling something about the actual preferences, so those should count much more than the browsings.

The mean is taken over the weights in equation 27, which is a decent way of aggregating when the weights are in a common range, since $w_i \in [0, 1]$ .

The above formula is very ad hoc, and only based on psychological arguments. Given the problem formulation requirements and the lack of data no other approach seemed possible. Nothing similar had been done in the past and no optimisation could be done to evaluate how large impact each of the weights had on the decision, that's why a simple average was calculated. Also the size of the factors mentioned are just guessed: it is almost certain that a selection should count more than a browsing, but we don't know how much more important it is.

## 6.3 Preference for food

The preference for a restaurant dish is calculated using the tf-idf measure as defined in equation 16. The algorithm works similar to Roccio's algorithm, by considering relevant and non-relevant menus, and calculating a menu query vector. Since the separation of dishes sometimes can be ambiguous, as explained in section 5.4, the whole menu text body of a restaurant is used when calculating the tf-idf descriptors.

When trying to figure out preferences for dishes, we take into account how long the user looks at a menu before continuing to the next, how many browsings the user made in a session compared to the mean number of browsings (how many restaurants that had to be browsed before the group made a decision) and how long back in time a restaurant browsing or selection occurred. Also it is taken into account if the operator returns to the menu of a restaurant several times.

**Browsing time** If a user immediately requests a new menu after looking at the dishes of a restaurant one can imagine that the dishes are perhaps not really interesting to the user. If on the other hand the user stays longer at a restaurant menu, it should mean that the dishes it includes could be more relevant. The first menu weight is defined as

$$v_1 = \begin{cases} \frac{t_{norm}}{t_{mean}} - 1 & \frac{t_{norm}}{t_{mean}} \leq 2 \\ 1 & \frac{t_{norm}}{t_{mean}} > 2 \end{cases} \tag{29}$$

where $v_1 \in [-1, 1]$ and $t_{norm}$ is the text-length normalised browsing time of a menu, and $t_{mean}$ is the mean text-length normalised browsing time of menus for a user as explained in the definitions above. The reason for normalising is that a long menu obviously will take longer time to read trough.

**Number of restaurants in browsing session** If the user on average checks out a $i_{mean}$ number of restaurants each day, it is reasonable to assume that something has happened if the number of restaurants in a browsing session, $i_{max}$ is drastically different to this mean on a particular occasion. If the operator stops looking at the restaurants after a short period of time, there is a high probability that the user or the group found dishes that everybody liked. If on the other hand the user continues to browse the menus, long after the mean has been reached, it is perhaps because the user can't find anything good. The weight capturing this is $w_2$ and is defined as

$$v_2 = \begin{cases} (1 - (\frac{i_{max}-i}{i_{max}})^2)(1 - \frac{i_{max}}{i_{mean}}) & \frac{i_{max}}{i_{mean}} \leq 1 \\ -(\frac{i_{max}-i}{i_{max}})^2(\frac{i_{max}}{i_{mean}} - 1) & 1 < \frac{i_{max}}{i_{mean}} \leq 2 \\ -(\frac{i_{max}-i}{i_{max}})^2 & 2 < \frac{i_{max}}{i_{mean}} \end{cases} \tag{30}$$

where approximately $v_2 \in [-1, 1]$. One should be reminded that $i$ and $i_{max}$ are discrete variables, where $i \in \{1, 2, 3 \ ... \ i_{max}\}$ The variable $i$ is the order a restaurant had in the browsing session where only its latest browsing is counted. The closer the number of browsed restaurants $i_{max}$, is to the mean $i_{mean}$, the smaller $w_1$ should be. The weights for the menus in the browsing session depending on $i$, $i_{max}$ and $i_{mean}$ are plotted in figure 11, 12, 13, 14, 15, 16, 17 and 18. Each red dot symbolises a restaurant and its menu. It can be seen that the weights are strongly positive if only a few menus have been read, and they start to become negative if the user continues looking at restaurants past the mean. In all example figures $i_{mean} = 10$.

Since only the latest browsing of a restaurant menu is counted, $i_{max}$ can never be larger than the total number of restaurants available in the list, $i_{max} \leq n_{tot}$. The reason for using second grade equations is that we assume that one of the last browsed menus is the actual selection for the user, which is perhaps satisfied with the menu. Because of this the latest browsed menus should have similar weights which will be the case around a quadratic maxima. Another argument is that a user should be well informed about the contents of the restaurant before selecting it. If the browsing occurred too long back in the browsing session, it is reasonable to think that the user checks back a last time on the selected restaurant, otherwise the contents might have been forgotten.

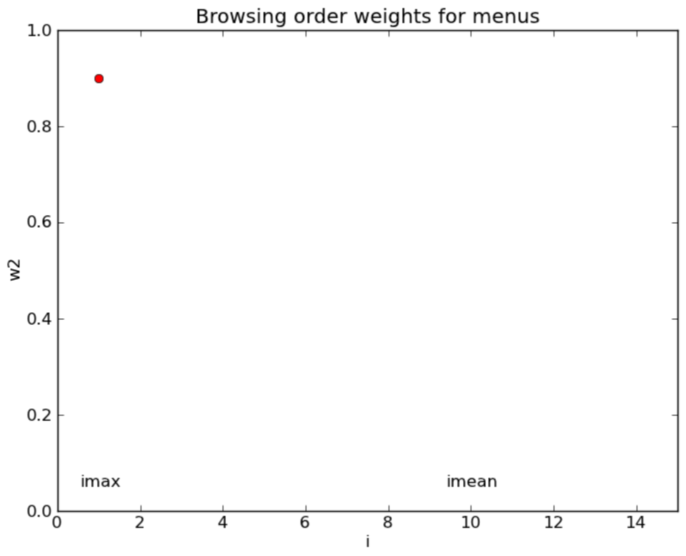


Figure 11: $\frac{i_{max}}{i_{mean}} \leq 1, i_{max} = 1$, the only browsed menu gets a high weight.

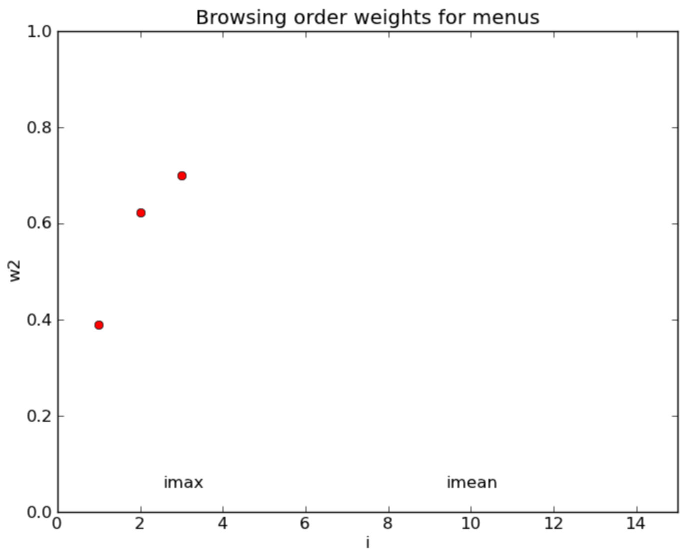


Figure 12: $\frac{i_{max}}{i_{mean}} \leq 1, i_{max} = 3$, there are several menus that get high weights

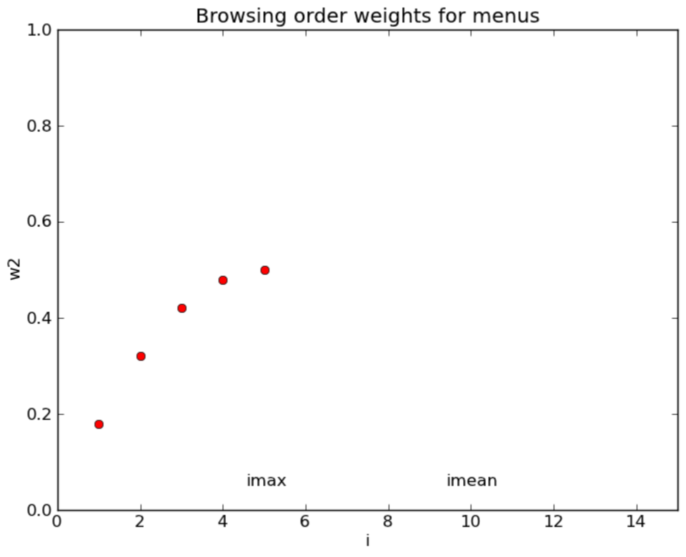


Figure 13: $\frac{i_{max}}{i_{mean}} \leq 1, i_{max} = 5$, the closer we get to mean, the lower the positive weigths.

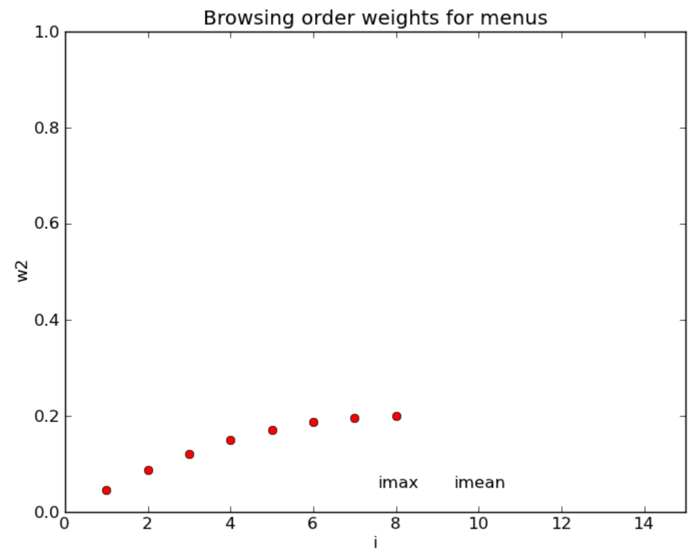


Figure 14: $\frac{i_{max}}{i_{mean}} \leq 1, i_{max} = 8$ the weights approaches zero. At mean they are all zeroed.

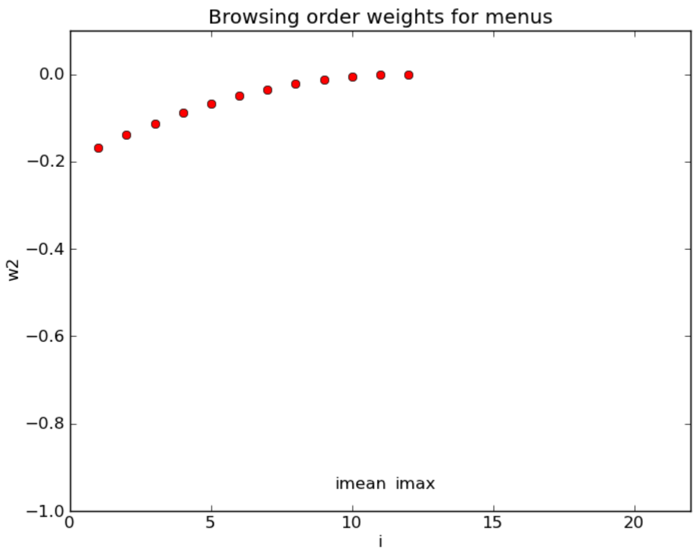


Figure 15: $1 < \frac{i_{max}}{i_{mean}} \leq 2, i_{max} = 12$, the weights start to become negative.

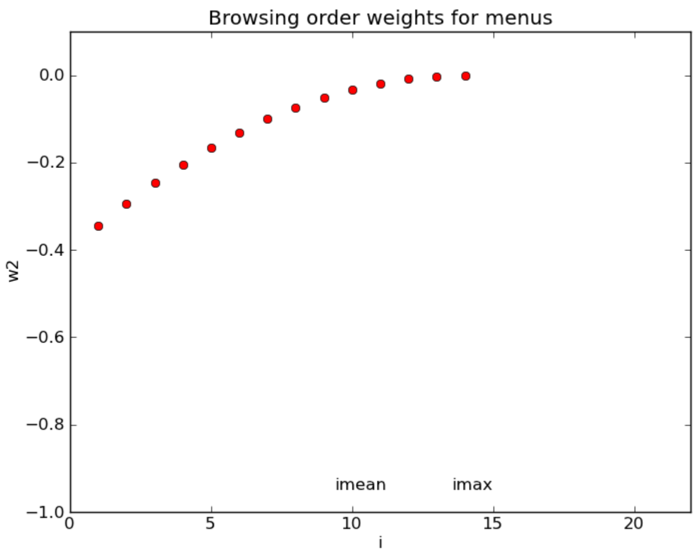


Figure 16: $1 < \frac{i_{max}}{i_{mean}} \leq 2, i_{max} = 14$, the last browsed restaurants are supposed to be the selection, they have almost zero weight.

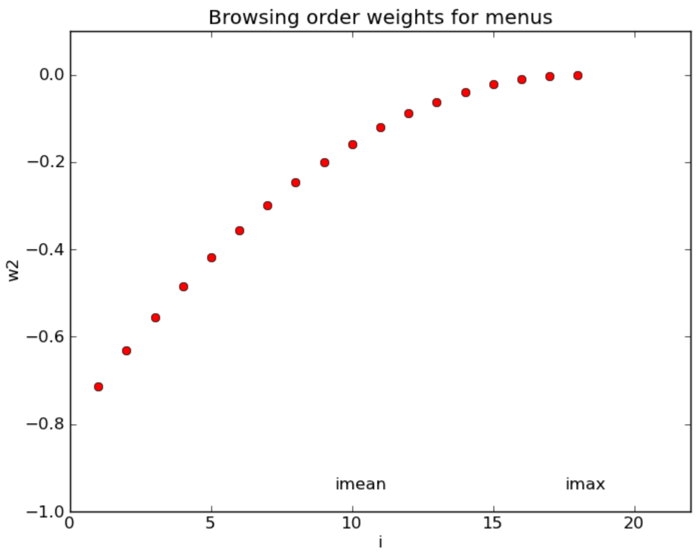


Figure 17: $1 < \frac{i_{max}}{i_{mean}} \leq 2, i_{max} = 18$, the weights decrease even more.

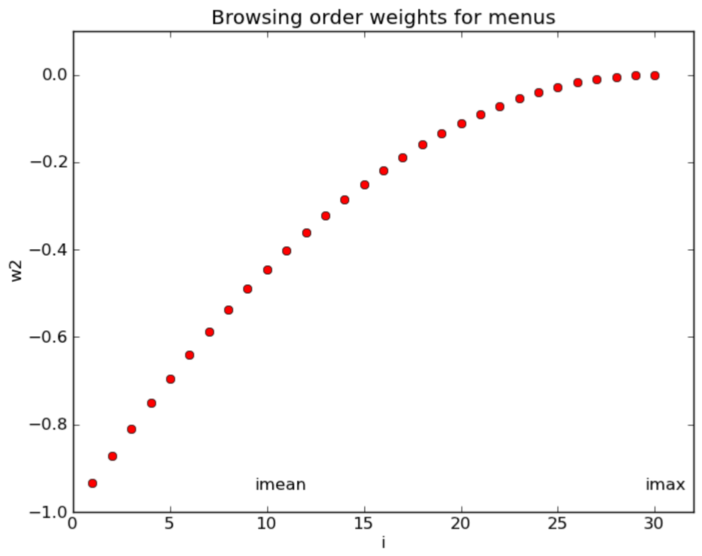


Figure 18: $2 < \frac{i_{max}}{i_{mean}}, i_{max} = 30$, we want $v_2 \geq -1$, that's why we have the last condition in equation 30. In this figure this condition is fulfilled.

**Time since browsing or selection** As with the restaurant preferences explained earlier, the ratings are assumed to change over time. The further back a browsing of a menu occurred in time, the lower importance it should have. $v_3$ is defined exactly as in equation 24.

**Rebrowsing** If a user checks back on a restaurant several times during a browsing session, the menu for that date should clearly be considered. Partly this rebrowsing behaviour already increases the weight for the food since the restaurant browsing order $i$ becomes lower, which means that $v_2$ increases. Additionally, several browsing occasions on a restaurant further increases the preferences for its menu, since the tf-idf vector are added several times to the user food preference vector, as we shall see later. Still it would be good to have a weight that alone captures this rebrowsing pattern.

$$v_4 = \frac{\sum\limits_{b \in U : restaurant(b) = i} \frac{j}{j_{max}}}{j_{max}} \tag{31}$$

A browsing is as before denoted $b$, and $U$ is the set of browsings which were in the browsing session of a user at a particular date in which $b$ occurred, and we have that $|U| = j_{max}$. This weight is calculated for the menu of restaurant $i$, so we only want to consider browsings of this restaurant, $restaurant(b) = i$. The discrete variables $j$ and $j_{max}$

are defined previously, and they are both depending on the browsing session, browsing, user and restaurant selected. If a user selects a certain restaurant many times towards the end of the browsing session this weight will become large. We also have that $v_4 \in [0,1]$.

**Group interaction** Exactly the same group dynamics can be argued to happen when calculating food preferences as with the restaurant preferences. Therefore $v_{interaction}$ again is defined as in equation 26.

**Calculating tf-idf profile vector** For each user $u$, a tf-idf food preference vector $\overline{q}_u$ is calculated.

$$\begin{aligned}\overline{q}_u = &\sum_{b \in B : operator(b)=u} \overline{d}_{restaurant(b)} \left(\frac{1}{2} + \frac{1}{2} v_{interaction}(b)\right) \frac{v_1(b)+v_2(b)+v_3(b)+v_4(b)}{4} \\ &\sum_{b \in B : u \in friends(b)} \overline{d}_{restaurant(b)} \frac{v_{interaction}(b)}{10} \frac{v_1(b)+v_2(b)+v_3(b)+v_4(b)}{4} + \\ &\sum_{s \in S : operator(s)=u} \overline{d}_{restaurant(s)} \cdot 10 v_{interaction}(s) v_3(s) + \\ &\sum_{s \in S : u \in friends(s)} \overline{d}_{restaurant(s)} \cdot 10 v_{interaction}(s) v_3(s) \qquad (32)\end{aligned}$$

As in equation 27, all the weights in the above expression are dependent on the actual browsing, selection, users and browsing session. Browsing time, order in browsing session and rebrowsing have no meaning when calculating preference for menu in a selection, that's why only $v_3$ may be used in the two last terms in equation 32. The tf-idf vector for a restaurant menu is denoted $\overline{d}_{restaurant(x)}$, where $restaurant(x)$ as before returns the restaurant of the browsing or selection $x$. Exactly the same arguments about the operators role in the browsing session, and the relative importance of a selection compared to a browsing, as laid out in equation 27 and explained by the group browsing assumption, are assumed in equation 32.

The different weights in equation 27 have different ranges, and the first two can become negative. The mean of these four weights as used in the equation can therefore take values between $-0.5$ and 1. This is not a problem, since the first two weights can actually, by psychological arguments described earlier, say something negative about the menu of a restaurant. On the other hand if the menu is rebrowsed only a few times or a browsing occurred a long time ago, this will never tell you anything negative about the menu.

The next step is to calculate a score for each restaurant $i$ and user $u$ based on the food served on that date. This score is denoted $r^2_{iu}$ to avoid mixing it with the score in equation

27. Every morning the food preference score is calculated for each user and restaurant, and the result is then saved for easy access.

$$r_{iu}^2 = \frac{\overline{q}_u \cdot \overline{d}_i}{|\overline{q}_u||\overline{d}_i|} \tag{33}$$

The score is derived using the cosine similarity measure, first mentioned in section 4.1.1. Menus that are similar to the users food preference vector will get high scores.

## 6.4 Global popularity

In addition to the restaurant and food preference, one final measure is weighted in: the global liking of a restaurant. It simply captures how popular a restaurant is among all users. If $\overline{g}$ is a vector where each component $\tilde{g}_i$ is the number of times the menu of the restaurant $i$ have been requested by all users, then we may calculate a popular score for the restaurant $i$ by normalising the vector.

$$r_i^3 = \frac{\tilde{g}_i}{|\overline{g}|} \tag{34}$$

As seen in the above equation, the rating is user independent, so $r_{iu}^3 = r_i^3$.

## 6.5 Aggregating results

Finally the rankings of the restaurants are aggregated by calculating single scores for the items. There are many ways to do this, for an extensive study see the article by D. He and D. Wu, "Toward a robust data fusion for document retrieval". [22] The selected method is *CombMNZ* stated in equation 36. This method have been used in the past, and have shown to give good results. [23, 22]

$$\hat{r}_{iu}^c = \frac{r_{iu}^c - \min_{i'}(r_{i'u}^c)}{\max_{i'}(r_{i'u}^c) - \min_{i'}(r_{i'u}^c)} \tag{35}$$

$$r_{iu} = \left(\sum_{c=1}^{3} \hat{r}_{iu}^c\right) \sum_{c=1}^{3} thresh(\min_{i'}(r_{i'u}^c)) \tag{36}$$

$$thresh(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{37}$$

In the above expressions $c \in 1, 2, 3$ refers to the score types: restaurant preference food preference, and restaurant popularity. In equation 35 the scores are first normalised and in equation 36 they are put together into an aggregated score. The second term in equation

36 is there to put more emphasise on to the items where nonzero ratings exist for all the rating types. The threshold function in equation 37 helps achieving that.

If there are no persons added to the lunch partner list, all the restaurants are ordered according to their respective scores from the individual recommendation. By now we are able to calculate an individual recommendation for a user.

## 6.6 Group recommendation

Next comes the main task for this master thesis: to use the collected data in order to obtain a group recommendation for a given group of people. Here two methods are used to obtain scores for an item introduced to a group, and these scores are then aggregated together.

### 6.6.1 Group aggregation

In this part the individual aggregated scores are once again aggregated into group scores for items using the average, least misery and mutual disagreement measure, earlier explained in section 4.4.2 equation, 19, 20 and 21. A final score is obtained by a linear combination of these values.

$$score^1(i,G) = w_1 \times avr(i,G) + w_2 \times lms(i,G) + w_3 \times (1 - dis(i,G)) \tag{38}$$

The group of people going together is denoted $G$. In the above formula all the weights are set to equal numbers, $w_i = 1/3$, and all functions are previously stated in section 4.4.2.

### 6.6.2 Support vector machine for group recommendation

If a selection exist for a group in the selection table, this information is used to train a Support vector machine in order to capture group dynamics. The input to the algorithm is the set of users that were in the group, and the output is the restaurant. It implies that all the restaurants are different classes, given a group of users it is a classification task to figure out where they will go. In fact we do not only want to obtain a classification, the main goal is to calculate a probability for each class. The machine learning library used, scikit-learn, does have built in support for multi-class SVM as well as a method for probability estimation of classes first proposed by John C. Platt, known as Platt scaling.[24][25] For more in depth information consider a textbook in the subject and the article by Platt.

**Feature extraction** The task is to figure out how to represent a group as a vector, and it should be based on a given a list of lunch groups and their respective members in the selection table. Assume that all users are nodes and edges between any two users symbolises whether they both have been in a group eating lunch together. It would be reasonable to think that all users in a connected graph could eat together. Given that there are 18 users, with internal IDs 1 to 18 shown in figure 19, assume that the following groups have had lunch together: $\{1,6,18\}$, $\{18,4\}$, $\{18,17\}$, $\{17,14\}$, $\{17,8\}$, $\{7,13,9\}$, $\{13,3\}$, $\{9,15,12\}$, $\{15,16\}$, $\{10,2,11\}$, $\{5,2,11\}$ and $\{5,4,2\}$. Any two of these sets which have an intersection, i.e. at least a number in common, is then combined into the same group. This is done iteratively until only a number of disjoint sets exits, called user clusters. This method is known as *set consolidation*. The groups mentioned above are combined into three such disjoint user clusters, $\{1,6,18,14,17,8\}$, $\{3,13,7,9,12,15,16\}$, $\{10,2,11,5,4\}$ shown in figure 19 as blue, green and red. For each of these graphs in the figure a SVM multi-class probability predictor is trained. The input is a binary vector of which people in the graph that were in the group. Each persons position in this binary vector is fixed after the group combination has performed. An example of how this feature vector creation would look like for each of the three different user clusters is shown in equation 39.

$$\begin{gathered}\{1,6,18\} \rightarrow [1,1,1,0,0,0] \\ \{9,5,12\} \rightarrow [0,0,0,1,1,1,0] \\ \{5,4,2\} \rightarrow [0,1,0,1,1]\end{gathered} \tag{39}$$

**Simulated selection** If a group has performed browsing, but there has been no message sent, and thus a restaurant selection does not exist, a selection is simulated. This is based on the same weighs that estimates the liking for a menu. Given a set of restaurants in the list, $i \in I$, the selected restaurant $i_{selected}$ is assumed to be

$$i_{selected} = \arg\max_{i} \quad w_1 + w_2 + w_3 \tag{40}$$

The estimated selection is written into a table in the database named *simulated_selection*. Both data from the simulated_selection and the selection table are used when doing the set consolidation and extracting user clusters, as well as when training their respective Support vector machines.

**Probability estimation** The input to the system is a group of people, $G$, which consists of the operator and the friends in the lunch partner list. A set of user clusters $G_1, G_2, ..., G_y, ...G_N$ has been extracted when doing the set consolidation. Some of the users in $G$ might not be in any of the user clusters, since the set consolidation is only performed in the train and save session each morning. however we still need to find out which of the clusters is most similar to $G$. The selected cluster $y_{selected}$ is decided by the following

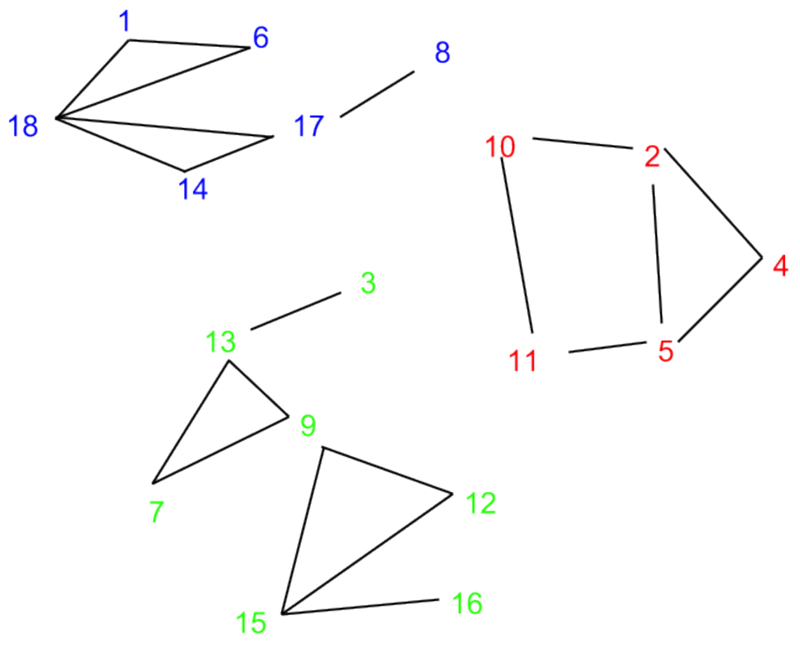


Figure 19: Network of lunch partners. Each number symbolises a person, and a connection between any two people means that they have eaten lunch together, either in a larger group or only the two of them. Different colours stand for different user clusters.

$$y_{selected} = \arg\max_{y} \quad |G \cap G_y| \tag{41}$$

After a cluster has been selected, the group is translated to a binary feature vector in a similar fashion as in equation 39. The group members that are not part of the cluster are simply left out. Finally the probability that the group $G$ will go to restaurant $i$ is calculated with the SVM, $prob(i, G)$. This is also set as the score.

$$score^2(i, G) = prob(i, G) \tag{42}$$

If there is only one person in the group, there could obviously be no group recommendation, and thus those examples are not considered both during training and group score estimation. Moreover if there is still too small intersection between the group and the selected user cluster, the probability prediction will not be very accurate. For that reason a threshold is set, $\frac{|G \cap G_y|}{G_y} > 0.5$. If this condition is not reached, no probabilities are returned from the Support vector machine method.

### 6.6.3 Aggregating group recommender results

If ratings exist for both equation 38 and 42 the scores are aggregated in exactly similar fashion as in equation 36. If scores for one of them is missing, if for instance there is too small intersection between the groups as explained above, only the score that exist will serve as group score for the item, without need to aggregate. If no information about the contacts in the lunch partner list could be extracted whatsoever, only the individual rating of the items from the operator will serve as group score. Finally the restaurants will be ordered according to the group score and we will have a group recommendation.

# Part III
# Results

After some delays the Android application was finally developed by the team in Beijing at the end of May. Then an extensive marketing campaign was carried out, where a thousand flyers were handed out in the area containing instructions how to download and use the application. In addition to this all the available restaurants in the application were visited, and most of them agreed to put leaflets in their restaurant.

Despite this effort, the reception of the application was modest at the best. Only a total of 47 users downloaded the application, and they performed in total 596 browsings and 42 selections, but only 473 of the browsings and 37 of the selections were used in the system, since a requirement was that browsing sessions or selections had to occur between 10 am in the morning to 15 pm in the afternoon, since these times are only relevant for having lunch. Out of the total number of browsings, 51 had lunch partners added, but only 28 of these group browsings were made at reasonable times. In addition to this, 16 selections were simulated. This means that quite a few selections were made without browsing at all, the users preferred to send the messages directly. But on the other hand it is hard to draw any conclusions from this, since all selections were made by only eight individuals.

The id_consumption_table contained 73 users, but only 5 of these had both information about IMEI-number and telephone number, which means that it at most occurred five contat-operator matches. But it is more likely that the users had their telephone number written into their Google profiles, and that this information was registered when the user opened the application. The total number of restaurants available was around 25, but that changed during the course of the project. However, since the code of the parser was robustly written this did't cause any problems.

# 7 Performance

Since the ratings are not explicitly collected for the food and the restaurants it is hard to find a way to measure the performance of the recommender system. In normal recommender system development the engineers can access a large dataset containing ratings by users for different items. This dataset is split into a training set and test set, and the trained algorithm tries to estimate the ratings of the items in the test set. By measuring the difference between the estimated and real ratings one can determine the performance. This can not be done in our case, since we have no definite ratings. Neither can the users be contacted to do some sort of poll where they would evaluate the system. However there is another way to get some sort of indication of how well the algorithm does the recommendation. The goal for the system is to order the items as accurate as possible, so that the user does not need to scroll down to find a good item. We propose that the higher position the browsed or selected items have on the list, the better is the recommendation algorithm performing. Moreover, if the average number of browsings in a browsing session tend to decrease, this should mean that the users find what they want faster, and thus increase user satisfaction.

## 7.1 Ranking of browsed restaurants

The first thing that was investigated is how the rankings of the browsed restaurants change over time. The results are shown in figure 20, 21 and 22. The total set of browsings by all users were analysed, and this set was divided in three parts depending on the timestamp of each browsing: from May 15 to June 5, June 5 to June 25 and finally June 25 to July 7. The ranking the browsed restaurant had in the list was recorded, and this is shown in the block diagrams. Our hypothesis was that after time passes and the algorithm learns the user's preferences, the browsings will be more concentrated to the top items in the list. However, it seemed that the opposite happened, the browsing ratio of the top items decreased over time. Still it is interesting to see that the assumption that a user is most likely to browse one of the top items in a list, regardless of personal preferences, as outlined in equation 25, section 6.2 seems to be valid.

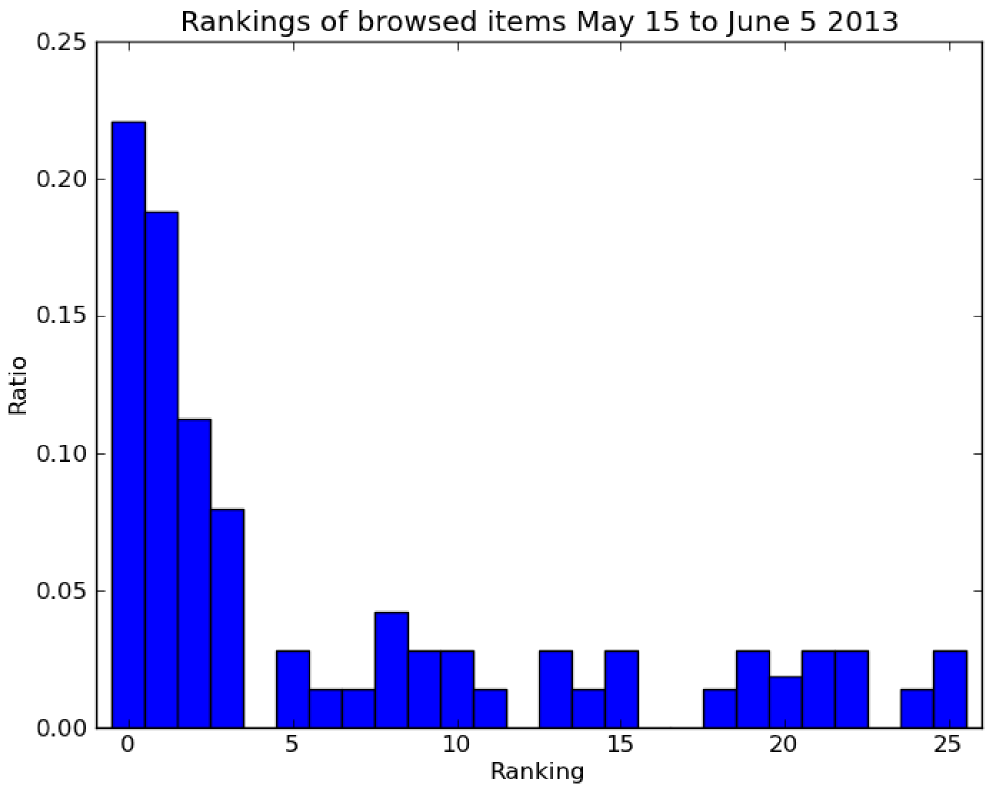


Figure 20: Rankings of the browsed restaurants by all users between the specified dates. The restaurant with zero ranking is highest up in the list, and the restaurant with 25 in ranking is at the bottom. The bars are normalised, so all the ratios will sum up to one. A total of 213 browsing requests were sent to the system during this time.

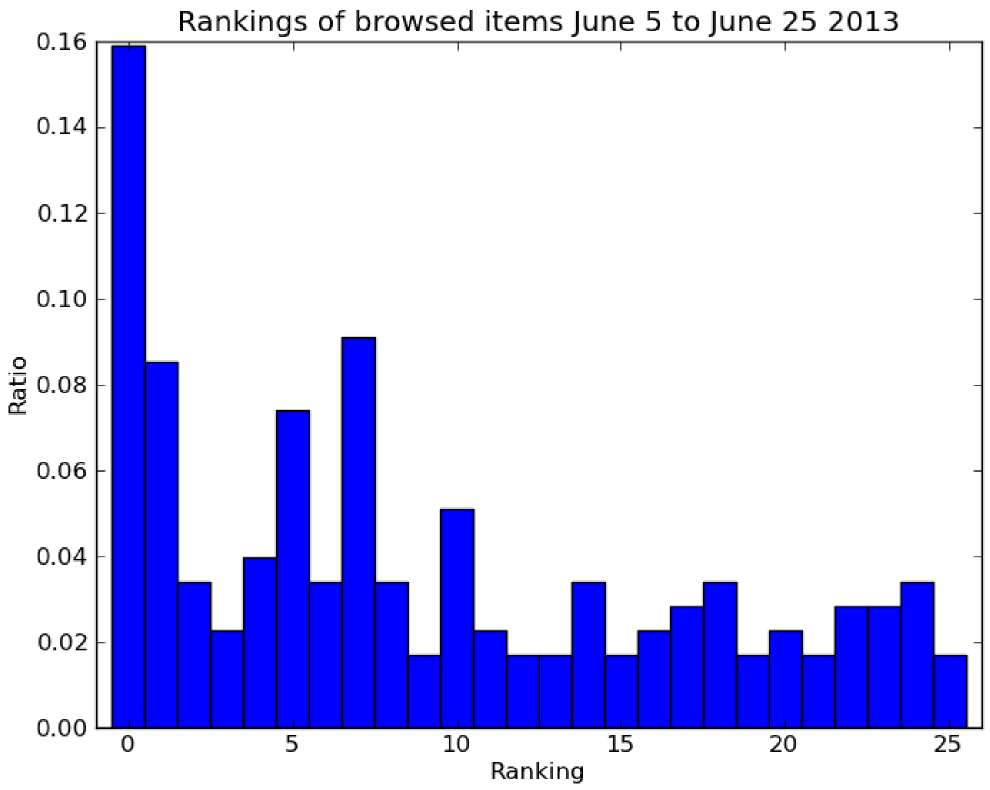


Figure 21: Rankings of the browsed restaurants by all users between the specified dates. The restaurant with zero ranking is highest up in the list, and the restaurant with 25 in ranking is at the bottom. The bars are normalised, so all the ratios will sum up to one. A total of 176 browsing requests were sent to the system during this time.

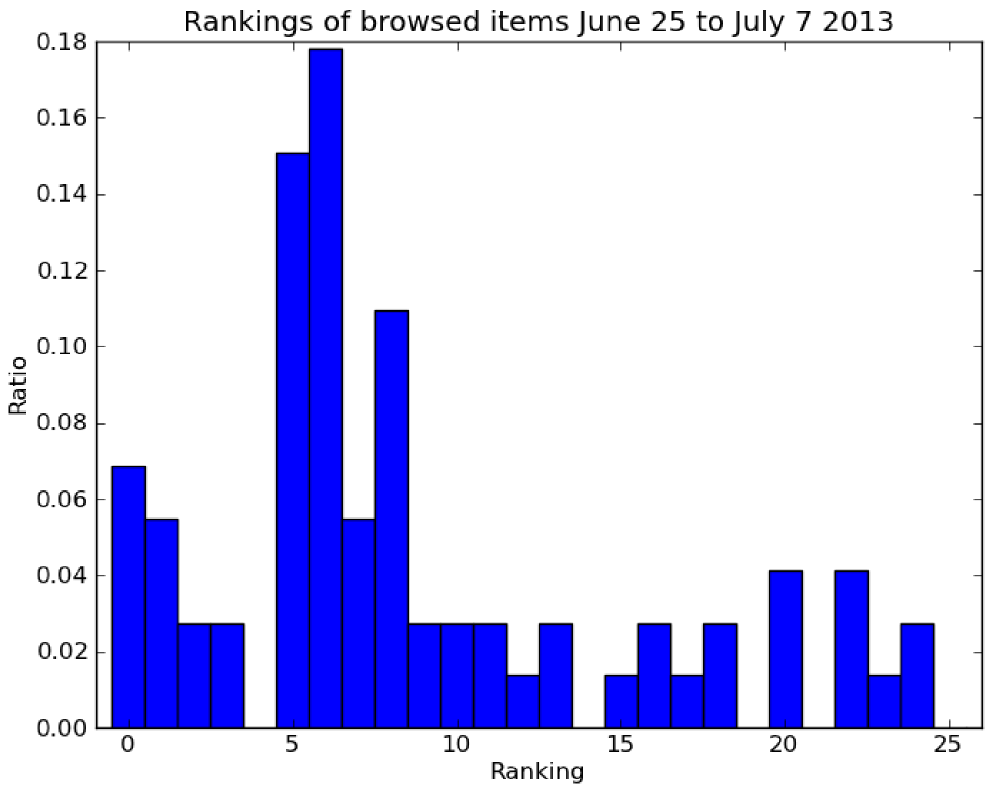


Figure 22: Rankings of the browsed restaurants by all users between the specified dates. The restaurant with zero ranking is highest up in the list, and the restaurant with 25 in ranking is at the bottom. The bars are normalised, so all the ratios will sum up to one. A total of 84 browsing requests were sent to the system during this time.

All types of browsings are included in these block diagrams, both those that had lunch partners added and those where the operator was browsing alone. Since so few used the group recommendation, the diagrams are mainly an indication of how the individual recommender system performed. From the middle of May a beta version of the application was used by a small group at Ericsson but in the end of May the full application was put online, and the vast majority of new users downloaded and started to use the service in the beginning of June. As we shall see later, due to the usage behaviour of the application no real conclusions can be drawn from the ranking of the browsed items.

### 7.2 Number of browsings in a browsing session

If the system performs satisfactory it should decrease the number of browsings a user or a group need to do before a decision has been made. After the browsing session is over, it is assumed that the group or individual user have come up with a decision. The mean number of browsings in a browsing session is shown in figure 23. This is based on all the browsing sessions by all user that occurred at each date. If only one operator performed a browsing session at one day, the number of browsed restaurants this session taken as the mean. From the diagram it is hard to see any improvements over time, but as we also shall see later, it is difficult to drawn any conclusions from the diagrams given the user data. An interesting phenomenon is that the weekends create gaps in the diagram, where no browsing sessions took place, and the mean is zero.

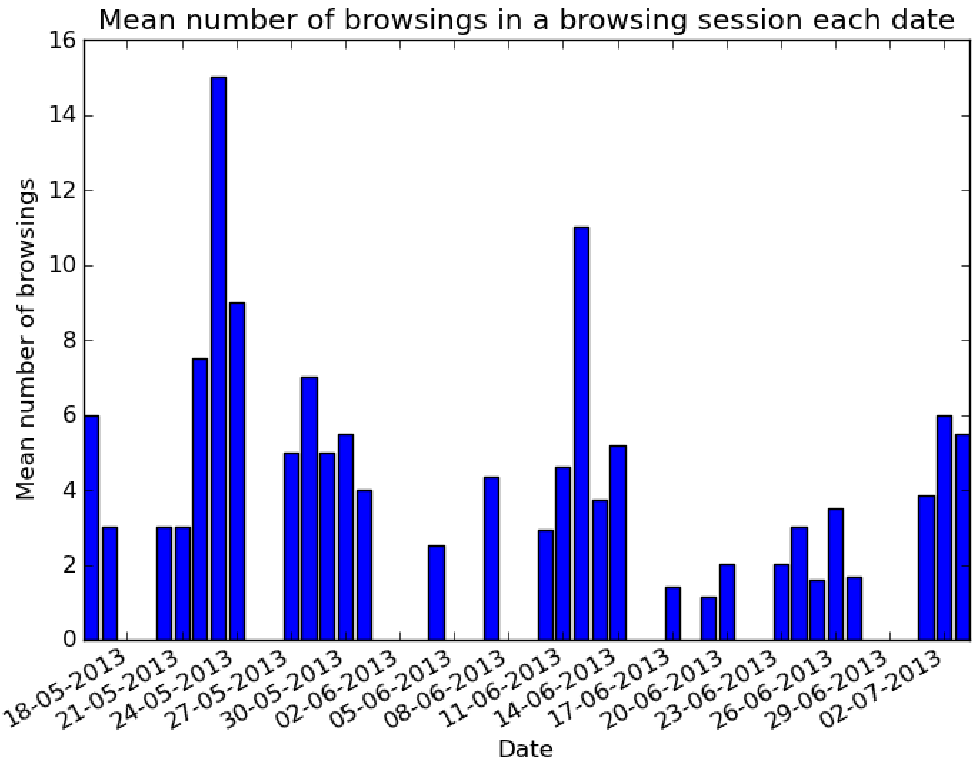


Figure 23: Number of browsings in a browsing session. The mean number of browsings in a browsing session is calculated at each date.

## 7.3 Rankings of selected items and group browsings

Due to the sparsity of group browsings and selected item recordings in the data base, divining these into subsets according to time and showing several bar diagrams, as done previously with all of the browsings, will not give any insights. The data will be even more sparse, and the plots will be of no use, so we can not track the performance of the group recommender algorithm over time. The figures 24 and 25 show the rankings of the selected and browsed items with people added as lunch partners. A selection can be only recorded in a group context, the two plots can be interpreted as the performance of the group recommender system.

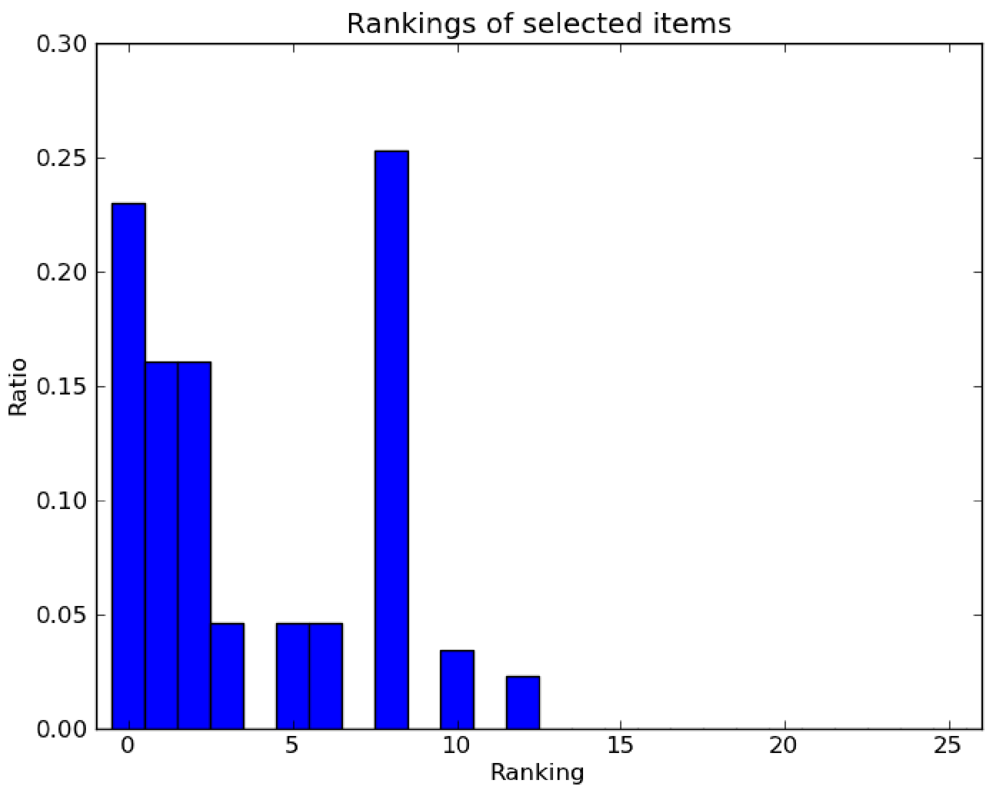


Figure 24: Rankings the selected restaurants had in the list. A total number of 37 selections were recorded in the database.

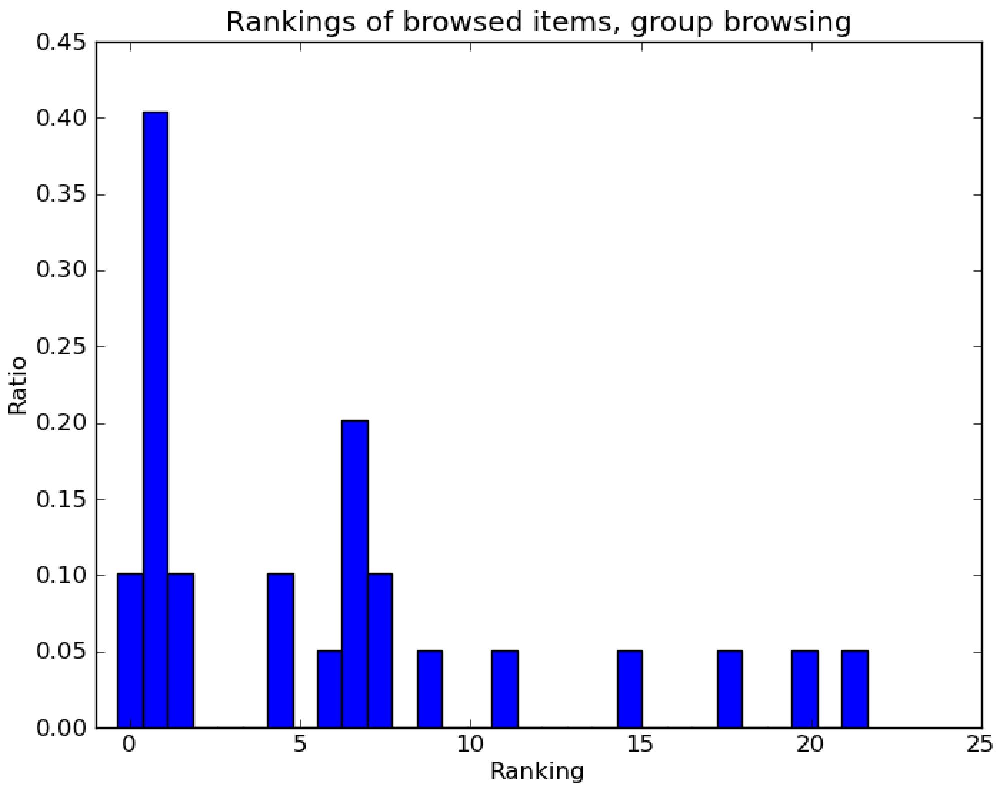


Figure 25: Rankings of the browsed restaurants with people added in the lunch partner list. A total number of 28 group browsings were recorded in the database.

From the diagrams we can see that the top ranked restaurants are more prone to selection, but apart from that no real insights can be made. There were in total eight individuals that made selections and only four people that tried to add lunch partners when browsing. This means that some users didn't bother to add lunch partners while browsing, that was done after a decision had been made, and the operator needed to tell a friend about the selection. However, it is uncertain whether such conclusions can be drawn, since the vast majority of the group browsings and selections were made by two employees in our team at Ericsson, the other individuals tested the functionality one or maximum two times.

## 8 User data

To understand the usefulness of the system and the recommender algorithm it can be interesting to analyse user data from the application. Figure 26 plots the number of browsing requests that have been sent to the server by all users over time. In the figure two spikes are clearly visible, the first happened in the end of May, and this is perhaps due to the beta version of the application, which was sent out to the research group via mail in the middle of May. The marketing campaign and handing out leaflets was carried out in the beginning of June, and its results can be seen in the middle of June, when the next big spike appears. It is apparent that there are a couple of days delay between the events and the spikes, it takes some time for the users to digest the message and install the application.

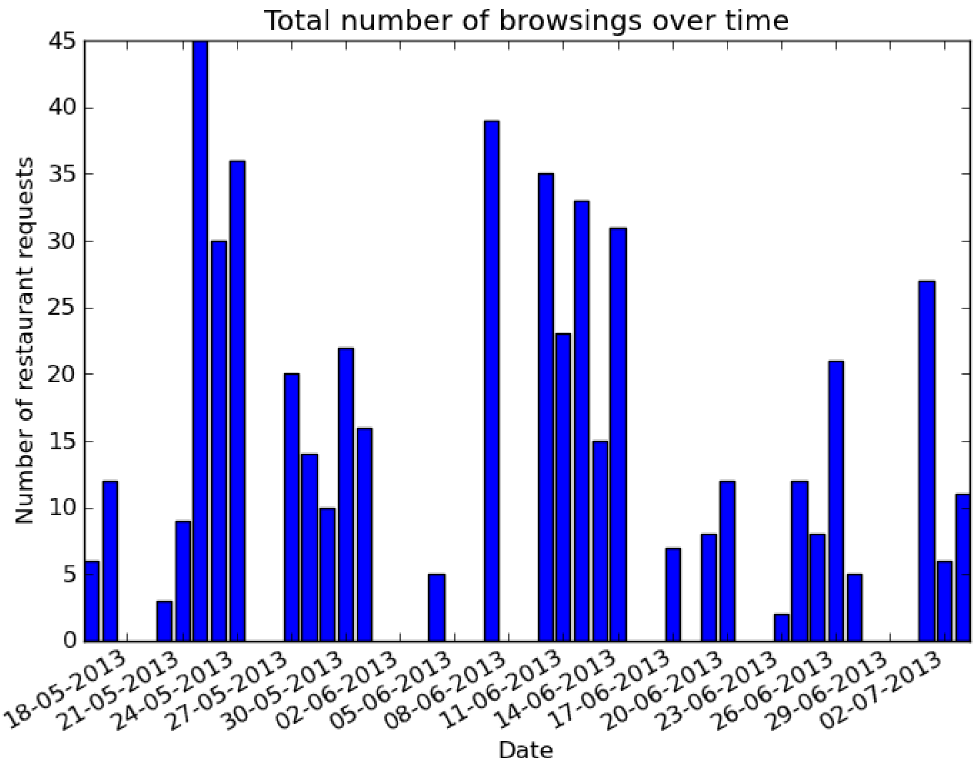


Figure 26: The total number of browsing requests sent to the server over time.

The public reception of the application was not very good, and this can be seen in how the browsings were distributed among the users. Figure 26 plots the total lifetime restaurant browsings by the users, and shows that the vast majority simply installed the application, did one browsing, and never used it again. Only two users, these were the two Ericsson employes involved in the project, did quite a few restaurant browsings, and these can be seen at around 80 and 110 browsings in the diagram.

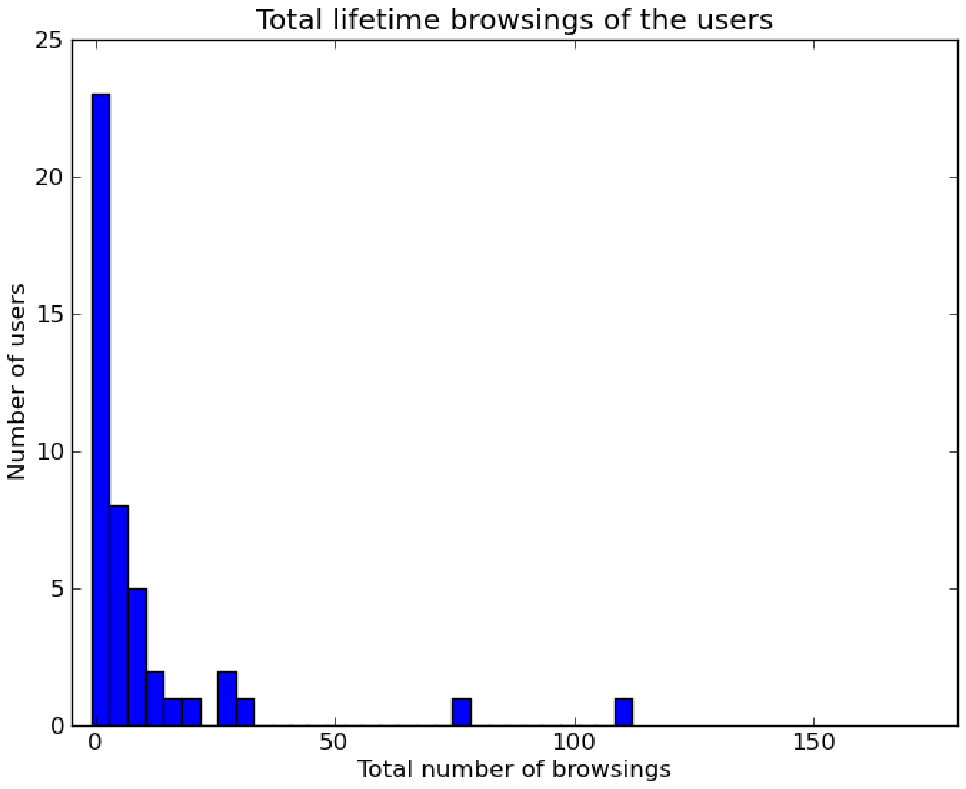


Figure 27: Distribution of the total number of restaurant browsings of the users.

It is also interesting to see for how many days the users consumed the application. Even though some users made relatively many browsings, perhaps they only used the application at one day. Since the recommender algorithm is trained once a day, there would have been no recommendation for these users. Figure 28 shows how many days the users used the application. Most users only used the application after it being installed, and then looked at one or two restaurants. The two Ericsson employes involved can once again be seen as the more frequent users.

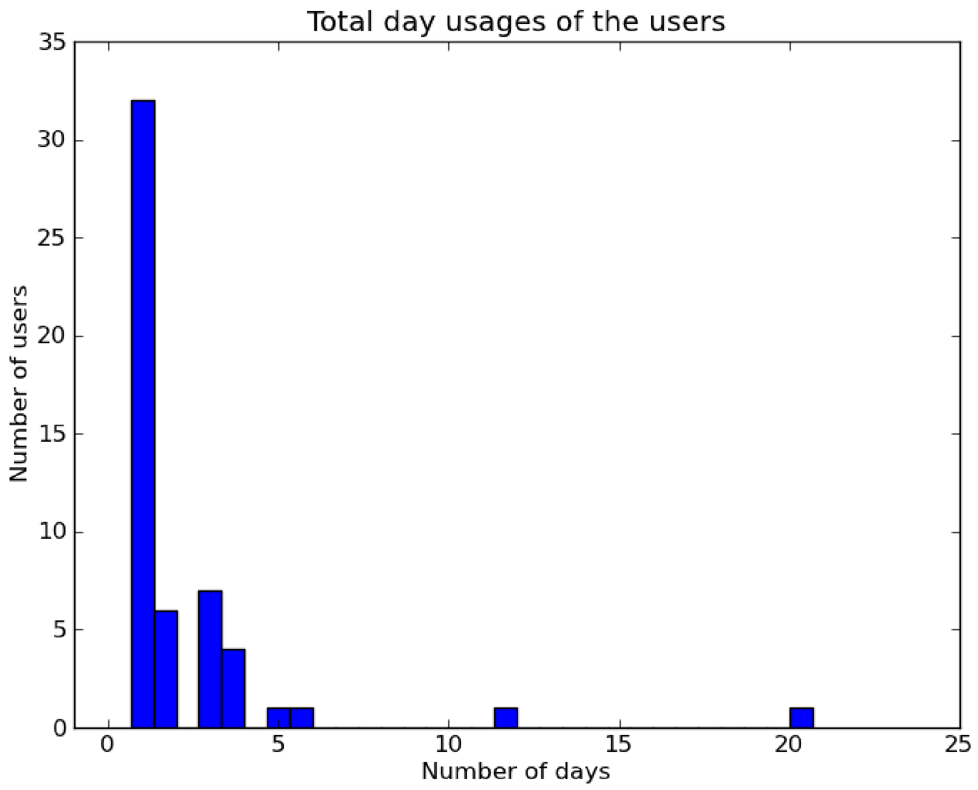


Figure 28: Distribution of the total number of day usages of the users.

As explained in earlier sections, the time between two menu requests from a user can be interpreted as the browsing time of an item. A threshold of 60 second was set on all browsing times, and they were saved for later analysis. The total distribution of browsing times for all users are shown in figure 29. These times are not normalised with the text length of the menus, which is the measure used in the recommender algorithm. It is clear that the highest probability for a restaurant browsing time, the mode, is around ten seconds, which is lower than the expected value or the mean, since the probability distribution is asymmetric and has a skewness. The mean is used in the recommender system, so this may cause too small weights, $w_2$ and $v_3$, capturing browsing time.

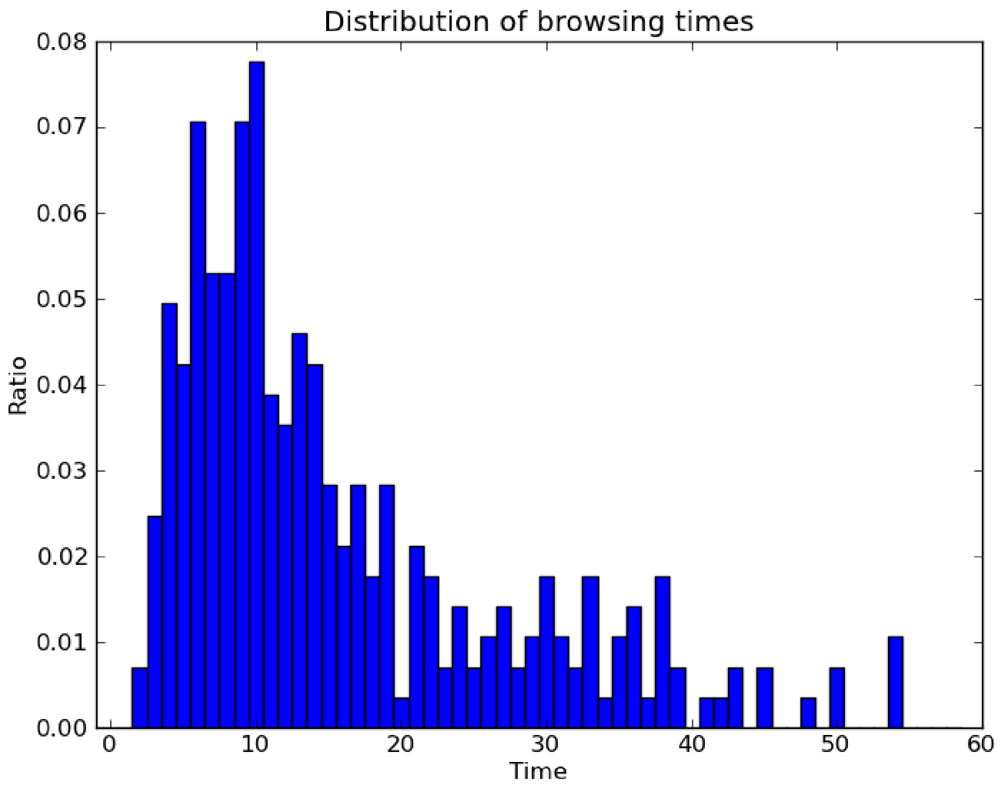


Figure 29:

## 9 Textual analysis of the menus

The content based functionality in the system estimates the users preference for the menus based on the menu text. As explained earlier the descriptors for this text were tf-idf feature vectors. Since the early days of this project, in the end of January, the parser was fully implemented and started to save menu texts to the database. Sometimes the menus were not explicitly written on the website, the text body just contained a link to the restaurant's homepage. Some chain store restaurants or food stands had the same menu every day, so their tf-idf vector did not change from day to day. It would be interesting to visualise the tf-idf feature vectors of the restaurant menus, and perhaps some conclusions can be drawn by understanding the textual data. The problem is that the vectors are 1000-dimensional, which means some dimensionality reduction technique needs to be employed in order to visualise the data. The Principal component analysis, PCA, is a standard method for doing this. The algorithm transforms the coordinate axes so that they are aligned in the directions with the largest variance in the dataset. The data points are then projected onto these axes, and only the two axes that correspond to the largest variance are selected for plotting. The results of the visualisation of the restaurant menus are shown in figure 30. In order to compare restaurants, the mean vector of the menus are calculated and shown in figure 31.

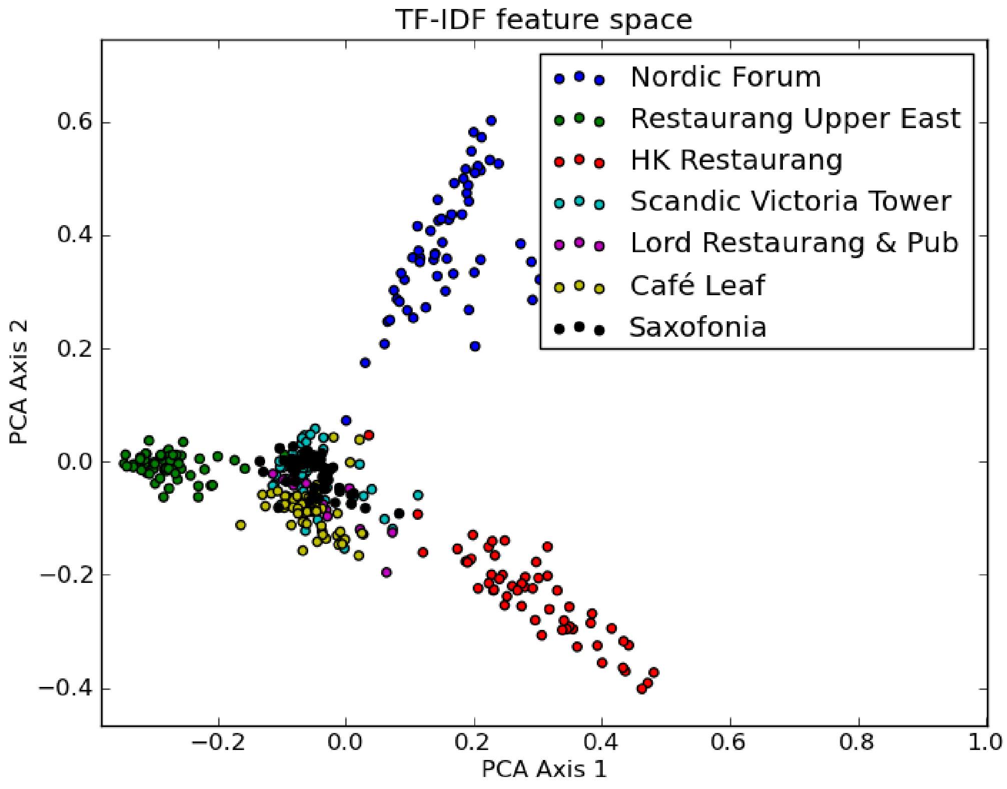


Figure 30: TF-IDF of menus, each point corresponds to the tf-idf vector of a restaurant's menu text body. Principle component analysis is performed and the vectors are visualised in 2D.

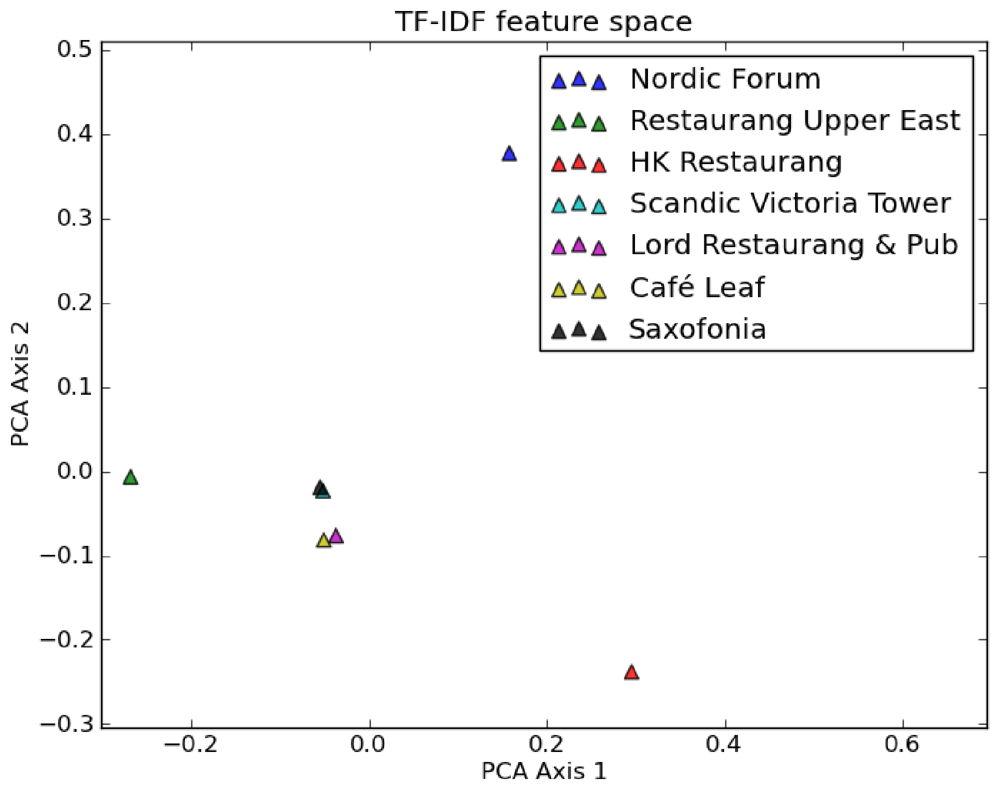


Figure 31: TF-IDF of menu centroids, each point corresponds to the mean of the visualised tf-idf vectors of a restaurant's menu text body shown in figure 30

Before the tf-idf vectors were calculated, stop words and a threshold for the minimum occurrence of words in the corpus were set. If a word only occurred once in the whole corpus, it was disregarded when calculating the tf-idf values. The selected stop words, that did not describe anything about the food content of the menus are shown in table 9

Another thing that can be analysed is how the occurrence of words in the corpus is distributed. Figure 9 shows all the words ordered by how frequent they are, and their respective total frequency in the whole document corpus. Around 1200 words were left after preprocessing was made.

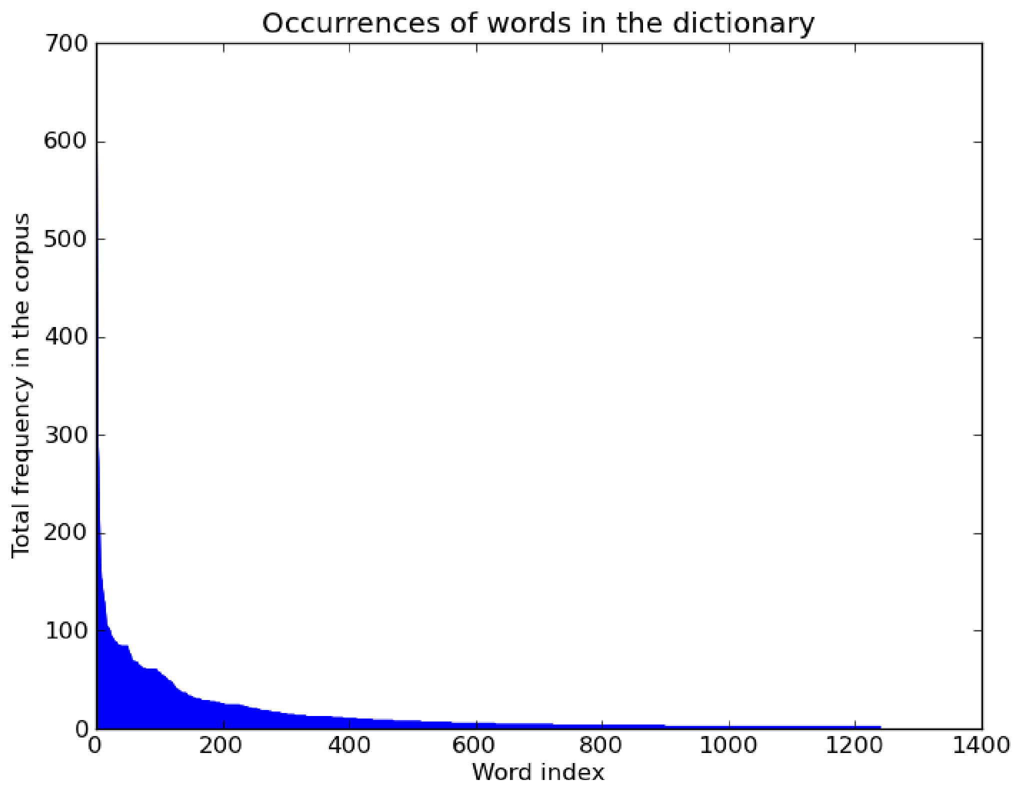


Figure 32: Frequency of all the words in the document corpus after stop words has been removed and minimum corpus frequency has been set.

Table 4: Frequency of words in the dictionary and selection of stop-words, shown in bold face.

| Word | Term frequency in corpus | English translation |
|---|---|---|
| **med** | 3174 | with |
| **och** | 1700 | and |
| **dagens** | 887 | today's |
| **på** | 863 | on |
| **meny** | 768 | menu |
| **att** | 684 | to |
| potatis | 650 | potatoes |
| **för** | 607 | for |
| **klicka** | 600 | click |
| **länken** | 600 | link |
| **se** | 600 | see |
| stekt | 570 | fried |
| kokt | 464 | cooked |
| **eller** | 374 | or |
| **samt** | 363 | and |
| ris | 288 | rice |
| vårrullar | 271 | spring rolls |
| kyckling | 219 | chicken |
| pasta | 195 | pasta |
| **av** | 169 | of |
| nem | 168 | nem |
| vegetarisk | 154 | vegetarian |
| **serveras** | 151 | served |
| ägg | 142 | eggs |
| lingon | 138 | lingonberry |
| potatismos | 131 | mashed potatoes |
| tomatsås | 129 | tomato sauce |
| **en** | 127 | one |
| persisk | 121 | persian |
| **kr** | 120 | crowns (currency in Sweden) |
| **som** | 120 | which |
| är | 115 | is |
| **hel** | 113 | whole |
| fläsk | 106 | pork |
| grädde | 105 | cream |
| sallad | 104 | sallad |
| panerad | 103 | panera |

Part IV

# Discussion and future work

## 10 Discussion

As can be seen from the figures presented that it is hard to see any obvious improvement over time. Most browsings are concentrated to the upper part of the list, but the relative concentration seem to change only slightly during the time the recommender system was in use. On the premise that an effective recommender system has been developed, is the assumption that most browsings will be concentrated to the upper part of the list valid? The author of this thesis would argue that this is the case. A user first checks out his or her favourite restaurants, and if the content is not enough satisfactory, the user will continue browsing. The users either trust that the content part of the recommender system will find suitable dishes, and don't want to spend the effort scrolling down, or they want to check out their favourite restaurants every day regardless of the menu. In either case it will lead to such weighting that the relevant restaurants will end up on the top of the list. Hopefully the content part of the system has selected dishes correctly, so that the user will be satisfied when viewing the content. Even though satisfied by the first restaurats and menus, some users may want to browse several other restaurants down the list. If the recommender system is working as expected, the content of these will be less suitable to the user, and the individual will eventually learn that only the top restaurants are relevant.

One issue that has not yet been discussed is that users might get tired of either dishes or restaurants. Clearly it is a flaw of the system if the user is recommended the same food everyday, despite the fact that it might be a favourite dish. However, it is unlikely that this will happen in the system, since every menu contains several dishes, and the ingredients can be found in many constellations. The food preference vector will have positive values for a lot of keywords, and it will tell what ingredients that are disliked rather than giving a high score to a particular dish. On the other hand, it may happen that the same set of restaurants always ending up at the top of the list, anyhow this is not necessarily a problem. A user may have a favourite lunch restaurant, where the menu is changed every day, and this will give enough variation. One should also be aware that the restaurants available on the website were of very different character. Everything from food stands, fast food restaurants and coffee houses to a la carte lunch restaurants appeared on the site. Also some restaurants were to far away and not relevant to the user. This would be apparent after some time since these restaurants would not get any menu requests.

When looking at the user data, it is apparent that the users did not really utilise the application. They simply installed it, made a few browsings, and then never touched it again. This can clearly be seen in figures 27 and 28. Because of this the results from the experiment are inconclusive, it is impossible to evaluate the performance of the recommender system. When studying the total browsing requests to the system in figure 26, it is not clear that they decrease, but this is due to the delay from the user seeing the leaflet to the installation of the application. New users dropped in continuously during the course of the experiment but none of them used the application consistently. The two peaks that can be seen in figure 26 are due to the marketing campaigns.

When examining figure 20, it appears that the system performed very well, even in the beginning when no ratings had been collected and no data was supplied to the training algorithm, and this is certainly a contradiction. The leaflets were mostly handed out to employees that worked in the area, and therefore often had lunch in one of the restaurants, so one could argue that they would already have opinions about the different places. Even so, they only requested to see the menus of restaurants at the top of the list, and didn't bother to find their favourite. If no ratings were collected, the list was simply in alphabetic order. This browsing behaviour shows that the users did not really try to use the application according to it's purpose, they simply tested it, by only pointing at the first items. The value they got from knowing the menus of the restaurants were not substantial, and perhaps not key for the decision.

There may be one flaw with the recommendation of dishes, which is based on textual analysis of the menus. As seen in picture 30 some of the restaurants clearly form clusters, even after the tf-idf vectors have been transformed from very high dimensions to only two dimensions. Because of the curse of dimensionality, it is possible that these clusters are even further apart in higher dimensions. The similarity between menus in the same restaurants may be caused by them serving special cuisine categories, for example Chinese, Indian, Italian etc, and thus uses special ingredients. However this is not the case for the restaurants in the figure. It may simply be that different individuals write the menus for the different restaurants and have their own way of expressing the dishes. This phenomenon may lead to the individual recommendation list just becoming a top list only ordered according to the number of browsings in each restaurant. When a user looks at the menu of a restaurant, the tf-idf food preference vector will increase accordingly, and the next day the menu of the very same restaurant will get the highest similarity. Another problem can be seen in figure 9, where it is apparent that most words only occur a few times in the menus. In total there where more than 4000 menus collected, and the most frequent words were only written a couple of hundred times, which means that relevant matches will seldom take place between menu tf-idf vectors. The vectors will be sparse and the similarities very small, and this is because the vocabulary in the menus being very specific, and the text lengths often very short. There are a lot of dishes out there that have funny names, and the chef may want to use a special vocabulary just to make the dish sound special and exotic. Perhaps two menus will have similarities only because they both contain "potatoes" or "rice", and this does not necessarily say anything about the users liking of the menu.

The author of this thesis is very sceptical about the group browsing assumption outlined in section 6.2 in the methodology part. It may be valid in other settings, where the group has to make an informed decision and the information is only provided in one device. But in reality only a few individuals require to see the menu before going to a restaurant with colleagues, and since a majority uses smartphones nowadays they can get the information from their own device. Since so few people used the group recommendation it is impossible to evaluate its performance. There are other apps and mobile web pages where the user can view multiple menus from different restaurants in one page, and more efficiently seek out suitable dishes. Also, since the nearby restaurants considered by the group were not that many, looking at their menus and coming up with a decision where to go is not a big deal to the users. The final decision will always be at the group, having an algorithm ordering the restaurants is perhaps not that useful. The users will discuss disadvantages, compromise and incorporate knowledge in the decision which is not known by the algorithm.

One other possible reason for the bad reception of the application is that it had some bugs and was not very well designed. Many users had to restart the application, sometimes it worked and sometimes it didn't. The development was heavily delayed, due to other assignments and lack of time, the main developer had to be changed during the course of the thesis. The bugs could not be found by the the development team, and they were slow to respond to corrections. It often took weeks before the team would make the required corrections.

# 11 Future work

It would be interesting to see the actual performance of the group recommender system, but this requires explicit information to be supplied by the users where the group went. Perhaps GPS technology could be used for this. Another thing that could be done is presenting the recommendation in a more fun way, showing the name of the restaurant, and telling the group the reason for the recommendation. Other knowledge could be used such as the weather, day of the week and the location of the restaurants. The recommender system then may display information about the restaurants through an interface, and for instance state that "persons A, B and C like this particular dish in the restaurant, the location is nearby and today the weather is bad". In that way the recommendation is more likely to be accepted by the group. If one tries to make the application fun and useful, perhaps some information has to be collected explicitly. The user could for example tell a number of dishes or ingredients that he or she likes and dislikes. Now we can only calculate the preference for whole menus based on implicit ratings, and this is obviously not very accurate. Also the input to the system is the preference after reading a description, it would be much better to have the user rate the item after the actual consumption has taken place. Finally synonyms could be used in a better way in the system, since most words occur so infrequently in the dictionary.

For the collection of implicit ratings, it is important to have the users browse trough restaurants, but this is not useful at all for the users. It would be much better showing the restaurants and menus on the same page. Experiments with eye tracking for implicit ratings collection could be made by using the front camera on the smartphones.

Since no collaborative approach has been used, it is unnecessary to have the system as a client server architecture. The users could browse restaurants on their own device, while the application collects the preferences. The group recommendation could be shown on all devices simultaneously by having the users turn on bluetooth and shake the phones together. The preferences of the friends would then be automatically transmitted.

# Part V
# Conclusions

Recommender systems are useful when having thousands of items and the user is unable to consider them all. The system developed in this thesis is ordering just a few restaurants in a list, and at the very best the value the users will get from the application is that they save a little time don't have to scroll or browse. The group recommender system is not a normal recommender system in the sense that it is trying to estimate ratings of items, it is trying to estimate a selection for a group. If there are only a few items available for selection it is not a cumbersome task for the group to make a decision, especially in this case when the users are familiar with most of the items. The value a group recommender system can give in this case is providing contextual information, telling the features of the items, and search the items for interesting and uninteresting content and thus helping the group to make an informed decision. Making such application is a programatically task, and perhaps machine learning do not need to be part of it. The final decision will always be made by the individuals in the group, and the goal should always be user satisfaction.

## References

[1] J. F. McCarthy, "Pocket restaurantfinder: A situated recommender system for groups," *Proceeding of Workshop on Mobile Ad-Hoc Communication at the 2002 ACM Conference on Human Factors in Computer Systems*, 2002.

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

[3] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems: An Introduction.* Cambridge University Press, September 2010.

[4] P. Melville and V. Sindhwani, "Recommender systems," *Encyclopedia of Machine Learning*, 2010.

[5] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," *The Adaptive Web*, pp. 325–341, 2007.

[6] K. Takano and K. F. Li, "An adaptive personalized recommender based on web-browsing behaviour learning," *IEEE Computer Society*, pp. 654–660, 2009.

[7] M. S. Pera and Y.-K. Ng, "A group recommender for movies based on content similarity and popularity," *Information Processing and Management: an International Journal*, vol. 49, pp. 673–687, May 2013.

[8] M. Balabanovic and Y. Shoham, "Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, no. 3, 1997.

[9] Y.-L. Chen, L.-C. Cheng, and C.-N. Chuang, "A group recommendation system with consideration of interactions among group members," *Expert Systems with Applications*, no. 34, pp. 2082–2090, 2008.

[10] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu, "Group recommendation: Semantics and efficiency," *Proceedings of Very Large Data Base Endowment*, vol. 2, pp. 754–756, August 2009.

[11] L. Boratto and S. Carta, "State-of-the-art in group recommendation and new approaches for automatic identification of groups," *Information Retrieval and Mining in Distributed Environments*, pp. 1–20, 2011.

[12] E. Ntoutsi, K. Stefanidis, and K. Norvag, "Fast group recommendations by applying user clustering," *Proceedings of the 31st international conference on Conceptual Modeling*, pp. 126–140, October 2012.

[13] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer Society*, 2009.

[14] R. Hastie and T. Kameda, "The robust beauty of majority rules in group decisions," *Psychological Review*, vol. 112, no. 2, pp. 494,508, 2005.

[15] A. Jameson and B. Smyth, "Recommendation to groups," *The Adaptive Web*, pp. 596–627, 2007.

[16] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl, "Polylens: A recommender system for groups of users," *ECSCW'01 Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work*, pp. 199–218, 2001.

[17] Z. Yu, X. Zhou, and J. G. Yanbin Hao, "Tv program recommendation for multiple viewers based on user profile merging," *User Modeling and User-Adapted Interaction*, vol. 16, no. 1, pp. 63–82, March 2006.

[18] S. Berkovsky and J. Freyne, "Group-based recipe recommendations: Analysis of data aggregation strategies," *RecSys'10 Proceedings of the fourth ACM conference on Recommender systems*, pp. 111–118, September 2010.

[19] L. Baltrunas, T. Makcinskas, and F. Ricci, "Group recommendation with rank aggregation and collaborative filtering," *RecSys'10 Proceedings of the fourth ACM conference on Recommender systems*, pp. 119–126, September 2010.

[20] J. Masthoff, "Group modeling: Selecting a sequence of television items to suit a group of viewers," *User Modeling and User-Adapted Interaction*, vol. 14, pp. 37–85, 2004.

[21] M. Claypool, P. Le, M. Wased, and D. Brown, "Implicit interest indicators," in *Proceedings of the 6th international conference on Intelligent user interfaces*, 2001, pp. 33–40.

[22] D. He and D. Wu, "Toward a robust data fusion for document retrieval," in *IEEE International Conference on Natural Language Processing and Knowledge Engineering*, 2008.

[23] J. H. Lee, "Analyses of multiple evidence combination," *SIGIR '97 Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 267–276, 1997.

[24] J. C. Platt, *Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods.* MIT Press, 1999.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.