Collaborative filtering with privacy

van Duijnhoven, A.E.M.

Award date:
2003

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computing Science

MASTER'S THESIS

Collaborative Filtering with Privacy

by
A.E.M. van Duijnhoven

Advisors:

Dr. Ir. C.A.J. Hurkens (Technische Universiteit Eindhoven)
Dr. Ir. W.F.J. Verhaegh (Philips Research)
Dr. J.H.M. Korst (Philips Research)

Eindhoven, 10th August 2003

Abstract

The growth of the world wide web has led to a situation where people are confronted with an overload of information. Collaborative filtering tries to relieve the problem of searching through all the information for interesting items. It recommends items to users by relating the preferences of the users to each other. In this report we study several collaborative filtering algorithms. Special attention is payed to the protection of the users preferences and protection of data valuable to the server. One algorithm, called factor analysis, has been implemented and tested with the Philips EasyAccess database. The preferences of the users and the data of the server can be protected with this algorithm. Factor analysis provides a good alternative to the popular user-based algorithm with Pearson correlation, as the prediction quality is just as good as the user-based algorithm, while more users can receive recommendations.

Preface

In this report, I give an account of my internship at Philips Research. The internship is my graduation project for the study Mathematics at the Technical University of Eindhoven. I wish to thank my advisors at Philips Research, Wim Verhaegh and Jan Korst; their ideas and suggestions have been a great help in writing this report. I am also grateful to Cor Hurkens, my thesis advisor at the TUE who helped me by giving comments on the text.

Eindhoven,
10th August 2003

Aukje van Duijnhoven

Chapter 1

Introduction

The explosive growth of the world wide web has led to a situation where people are confronted with an overload of information. In order to relieve the problem of searching through all the information for interesting items, a wide range of systems is being developed. These systems can be divided into three groups: information retrieval systems, information filtering systems and recommendation systems.

Information retrieval systems allow users to express queries to select documents that match a topic of interest.

Information filtering systems use the same techniques as information retrieval systems, but are meant for a stream of incoming documents. In this case the user has a long-term interest in certain topics.

Recommendation systems try to make a choice for a user based on his likes and dislikes.

An advantage of recommendation systems is that they can surprise the user with new, unknown items that he presumably likes. There are two kinds of recommendation systems:

Content-based systems use the content of the items and the user's preferences in the past to make a choice for the user.

Collaborative filtering systems recommend items to a user based on preferences of other users.

Collaborative filtering has some advantages over the other method. Subjective attributes such as quality, clarity and presentation style can be taken into account, since the system uses knowledge of other people who have accessed the item rather than properties of the content. As analysis of the content of an item is not necessary, the system can handle any type of data. However, collaborative filtering has some disadvantages too.

- Users must give their preferences about a lot of items before the system will work.
- New items can only be recommended after some users have evaluated them.
- Popular items, such as songs from The Beatles, are more often recommended.

A collaborative filtering system, called Jukebox, has been developed at Philips Research. This system is described in Section 1.1. The Jukebox system still has some flaws, which are described in Section 1.2. These flaws lead us to requirements for a new system, which are given in the problem statement (Section 1.3). We conclude this chapter with an outline of the report in Section 1.4.

1.1 Jukebox system

The Jukebox recommender [14, 15] is a collaborative filtering system that recommends songs to users. The users have a music player (jukebox) at home, with which they can listen to certain songs. When the users have listened to a song, they can describe their taste by rating this song via the interface of the jukebox. The songs are rated on a scale from 1 to 5. A rating (or vote) of 1 indicates that the user dislikes a song while a rating of 5 means that the user likes the song very much. The jukeboxes of the different users are all connected to a computer (server) which calculates the recommendations. For this purpose the server needs the ratings of the users. When a user makes a rating, the jukebox sends this rating via the connection to the server. The server stores the ratings in a large database (Figure 1.1).

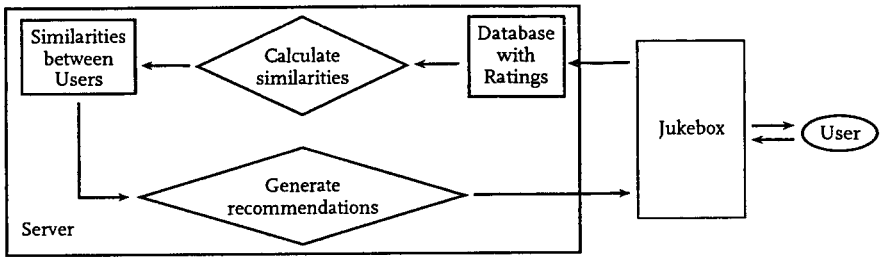

Figure 1.1: A schematic view of the Jukebox recommender system.

Computing similarities between users. We first introduce the terms profile and active user. A profile is the list with the songs the user rated combined with the ratings the user gave to the songs. The active user is the user for whom we want to make recommendations. If the active user wants a new song to listen to, the system searches in the database with all profiles for those user profiles that match the profile of the active user. In the Jukebox system the kappa statistic [23, 26] is used as a measure for matching (or similarity) between two users. The kappa statistic takes values between 0 and 1, where a 0 indicates that there is no matching at all, while a 1 indicates a perfect match. The kappa statistic is described in more detail in Section 2.2.1.

Generation of a recommendation list. To generate a recommendation for an active user, the following steps are performed.

1. All kappa coefficients for the active user are computed.
2. The coefficients that are lower than a certain threshold are ignored. These coefficients correspond to users with a taste different from the active user.
3. Select the user that corresponds to the largest coefficient. The profile of this user is the most similar to the profile of the active user.
4. Add all songs with a topscore (4 or 5) that the active user did not vote on to the list of recommended songs. Increment the number of votes by one for each song added to the list. Proceed with the user with the next best match.
5. Recommend the songs with the highest number of votes first. Most of the users similar to the active user like these songs.

The similarities between the users are not calculated after every update, because this is a lot of work. In the Jukebox system it was decided to calculate the similarities once a week.

1.2 Issues in collaborative filtering systems

The recommendations made with a standard collaborative filtering system (such as the Jukebox system) are quite good, but the system still has some flaws.

- The server stores the ratings of the users. This is very personal information that should be protected against unwanted use. The server extracts information from the data the user sends, such as relationships between certain songs. This information is used by the server to offer valuable services. For that reason, information valuable to the server should be protected too.
- The Jukebox algorithm requires a computation that grows with the number of users and songs. In the next chapter we shall see that the time complexity for the Jukebox algorithm is quadratic in the number of users. This means that if the number of users doubles, the computation time will be multiplied by four.
- The correlation measure (or kappa statistic) calculates the similarities based on songs the users both rated (Section 2.2.1). It is difficult to find users who rated the same songs as the active user, as there are a lot of songs and even a user who listens very often can not listen to all of them. This means that the correlation measure or the kappa statistic is very unreliable, as it is based on only a few songs. A lot of users are therefore not able to receive recommendations.

1.3 Problem statement

We assume that a new to build system has one server available, which is connected with the music players of the users. The music player could be the internet radio player Streamium, which is developed at Philips Research. Of course, there exist a lot of users with such a player, therefore we demand that the system can cope with a lot of people, for instance 10,000. For the security of the user information and the server information, we derived the following requirements.

- A user may not know the rating of any other (anonymous) user for a given song.
- A user may not know which songs any other (anonymous) user rated.
- A user may not know any data valuable to the server, such as dependencies between songs.
- The server may not know the rating of any user for a given song.
- The server may not know which songs any user rated.
- The server may not know which user resembles any other user.
- The server determines how many songs are recommended. If the user gets all the recommendations at once, he would use the recommender only one time, and the server would not make any profit.

Finally, the quality of prediction should be at least as good as the quality of prediction of the Jukebox system.

1.4 Outline of the report

In the next chapter an overview of collaborative filtering algorithms from the literature is given. Chapter 3 introduces the cryptographic techniques we use in order to secure the user profiles and the data valuable to the server. In the Chapters 4,5, and 6 we derive security protocols for the algorithms given in Chapter 2. We implemented the factor-analysis algorithm and tested it with the Philips EasyAccess database. This database is the result of an earlier experiment with the Jukebox system. The results of the test with the factor-analysis algorithm can be found in Chapter 7 and 8.

Chapter 2

Collaborative Filtering Algorithms

Collaborative filtering algorithms are usually divided into two categories, memory-based algorithms and model-based algorithms.

> *Memory-based algorithms* use the database with votes to calculate 'distances' between users. New predictions are obtained by interpolation. The memory-based algorithms can be subdivided into two categories, being user-based and item-based algorithms. The user-based algorithms are described in Section 2.2, while the item-based algorithms are described in Section 2.3.
>
> *Model-based algorithms* use the database with votes to build a model, which is then used for calculating predictions. An example is the factor-analysis algorithm, which is described in Section 2.4.

The next section describes our notation, and introduces an example which is used throughout the text in order to clarify the algorithms.

2.1 Notation

Suppose a tea and coffee company wishes to sell new flavours to their customers. The company sells three tea flavours (T1,T2,T3) and six coffee flavours (C1,...,C6). Via an enquiry it obtained the opinion from its customers for the flavours they already tasted, as shown in Table 2.1.

Taste	T1	T2	T3	C1	C2	C3	C4	C5	C6
Wim	2	1	1	4		4	3	4	3
Jan	1		1	5	5	4	4	3	
Arnout	4	5	5	2		3	3	2	
Aukje	5	4		1	3	2		2	

Table 2.1: Customer opinions of tea and coffee flavours.

In the enquiry a 1 was considered as a strong dislike for the flavour while a 5 meant that the user (customer) liked the flavour very much. We shall call these numbers ratings or votes. A list of items (in this case flavours) with the corresponding votes from a certain user is called the profile of the user. The company wants its customers to try tea or coffee they never tasted. However, the company only wants to do this if the customer would probably like its product. A collaborative filtering algorithm is used to make predictions for the missing votes. The missing flavour is recommended to the customer if the prediction for the vote is high enough. In the next sections we discuss how predictions can be made, for which we will use the notation as given in Figure 2.1.

Symbol	Description
U	number of users
I	number of items
x, y	users
a	active user
i, j	items
v_{xi}	vote from user x for item i
$\bar{v}_x$	mean vote of user x over his rated items
$\mathbf{v}_x$	vector with votes from user x, where a 0 indicates that an item has not been rated
r_{xi}	is 1 if user x rated item i and 0 otherwise
$\mathbf{r}_x$	vector with elements r_{xi}
I_x	set with items user x rated
U_i	set of users who rated item i
s	similarity
p_{ai}	prediction for user a and item i

Figure 2.1: Notation.

2.2 User-based algorithms

User-based algorithms are probably the oldest and most widely used collaborative filtering algorithms [3, 15, 18, 12, 19]. An example of a user-based algorithm is the one used in the Jukebox system. First a similarity measure is calculated between every pair of users, indicating to what extent their profiles match. Next, a prediction for item i is calculated by taking a weighted average over the users who gave their preference about item i, where the similarities between users define the weights. If users have a high similarity with the active user, their influence in the predicted vote is bigger. There are a lot of possibilities for the choice of the similarity measure and the way in which we make predictions, as we show in Sections 2.2.1 and 2.2.2. For the similarity measure we can for instance choose distance metrics, metrics based on counting the number of items both users like or correlation metrics. The one that is most often used is

$$s(x,y) = \frac{\sum_{i \in I_x \cap I_y} (v_{xi} - \bar{v}_x)(v_{yi} - \bar{v}_y)}{\sqrt{\sum_{i \in I_x \cap I_y} (v_{xi} - \bar{v}_x)^2 \sum_{i \in I_x \cap I_y} (v_{yi} - \bar{v}_y)^2}}, \tag{2.1}$$

called the Pearson correlation, and the corresponding prediction for user a and item i is given by

$$p_{ai} = \bar{v}_a + \frac{\sum_{y \in U_i} s(a,y)(v_{yi} - \bar{v}_y)}{\sum_{y \in U_i} |s(a,y)|}. \tag{2.2}$$

Not only users with a high correlation influence a prediction for the active user in (2.2). Users with a very negative correlation have a taste opposite to the active user, so if a user with a negative correlation likes an item, the active user will probably dislike it.

Example 2.1: Suppose Aukje wants to know her expected preference for item T3. We start calculating the similarities between Aukje and the other users with the Pearson correlation (2.1). The mean $\bar{v}_{\text{Aukje}} = \frac{5+4+1+3+2+2}{6} \approx 2.8$. The mean of user Jan can be calculated in the same way, $\bar{v}_{\text{Jan}} \approx 3.3$. The set $I_{\text{Aukje}} \cap I_{\text{Jan}} = \{\text{T1}, \text{C1}, \text{C2}, \text{C3}, \text{C5}\}$. If we fill in the proper values, we obtain

$$\begin{aligned} s(\text{Aukje,Jan}) &= \frac{2.2 \cdot (-2.3) - 1.8 \cdot 1.7 + 0.2 \cdot 1.7 - 0.8 \cdot 0.7 - 0.8 \cdot (-0.3)}{\sqrt{2.2^2 + 1.8^2 + 0.2^2 + 0.8^2 + 0.8^2}\sqrt{2.3^2 + 1.7^2 + 1.7^2 + 0.7^2 + 0.3^2}} \\ &\approx \frac{-8.1}{3.1 \cdot 3.4} \approx -0.77 \end{aligned}$$

A similar calculation can be performed for the other pairs of users, resulting in similarity values as shown in Table 2.2. Note that the correlation is symmetric, i.e. the correlation between Jan and Aukje is the same as the correlation between Aukje and Jan.

	Wim	Jan	Arnout	Aukje
Wim				
Jan	0.78			
Arnout	-0.96	-0.74		
Aukje	-0.85	-0.77	0.83	

Table 2.2: The Pearson correlation between the users in the example of Table 2.1.

If we use predictor (2.2) then the prediction for Aukje and T3 is

$$2.8 + \frac{-0.85 \cdot (-1.8) - 0.77 \cdot (-2.3) + 0.83 \cdot 1.6}{0.85 + 0.77 + 0.83} \approx 4.7.$$

This means that Aukje will probably like tea T3.

If we want predictions for all missing votes, then we have to calculate $O(U^2)$ similarities. Every similarity consists of three sums over the items. Therefore the similarity-calculation phase will cost $O(U^2 I)$ time. The prediction phase has the same time complexity: every user wants $O(I)$ predictions and every prediction consist of a sum over the users. The total algorithm thus has a time complexity of $O(U^2 I)$.

2.2.1 Similarity measures for the user-based algorithm

We mentioned earlier that there are a lot of similarity measures. The similarity can be a distance between two profiles, the correlation or a measure of the number of equal votes between two profiles. For the calculation of the predictions, it is necessary that the similarities are high if the users have the same taste, and low if they have an opposite taste.

Distance measures

The distance calculates the total difference in votes between the users. The distance is zero if the users have exactly the same taste. The distance is high if the users behave totally opposite. Therefore we have to do an adjustment such that the weights are high if the users vote the same. A simple distance measure is the Manhattan distance [1, 11], which is given by

$$\frac{\sum_{i \in I_x \cap I_y} |v_{xi} - v_{yi}|}{|I_x \cap I_y|}. \tag{2.3}$$

Another distance measure is the mean-square difference [25], which is based on the squared difference between the votes. The mean-square difference is given by

$$\frac{\sum_{i \in I_x \cap I_y} (v_{xi} - v_{yi})^2}{|I_x \cap I_y|}. \tag{2.4}$$

Example 2.2: The Manhattan distance between Wim and Aukje is

$$(|2-5| + |1-4| + |4-1| + |4-2| + |4-2|)/5 = 2.6.$$

The Manhattan distance between Arnout and Aukje is 0.8. The maximum possible distance is $5-1 = 4$ while the minimum distance is 0. To make proper weights, we adjust this distance by subtracting the

real distance from the mean distance 2. The weight between Wim and Aukje is hence $2 - 2.6 = -0.6$ while the weight between Arnout and Aukje is $2 - 0.8 = 1.2$. We can calculate the mean-square difference between Wim and Aukje in the same way. This distance is equal to 7. The mean-square difference between Arnout and Aukje is still 0.8. The mean-square difference takes values between 0 and $(5-1)^2 = 16$. To make proper weights, we subtract $2^2 = 4$, where 2 is the mean difference between the votes. If we took 8, the weight between Wim and Aukje would be positive, indicating that they are similar. The weight between Wim and Aukje is $4 - 7 = -3$ while the weight between Arnout and Aukje is $4 - 0.8 = 3.2$. The mean-square difference discriminates better between the users with the same taste and users with an opposite taste. However, users with an opposite taste receive high negative weights, as the weights are between -12 and 4.

Correlation measures

We have already seen the Pearson correlation (2.1). We mentioned that a high correlation (close to 1) indicates that the users have similar profiles, while a low correlation (close to -1) indicates that users have an opposite taste. However, as the correlation is actually a measure of the linear relationship between two users, this is not always true. For instance, if user x votes 1 for all items and user y votes 5 for all items, then the correlation between the users x and y is 1. When we use the Pearson correlation as similarity measure, we make the assumption that the means of the users are approximately the same. The Pearson correlation has a variant called the constrained Pearson correlation [14], which does not have this problem, and which is given by

$$s(x,y) = \frac{\sum_{i \in I_x \cap I_y} (v_{xi} - c)(v_{yi} - c)}{\sqrt{\sum_{i \in I_x \cap I_y} (v_{xi} - c)^2 \sum_{i \in I_x \cap I_y} (v_{yi} - c)^2}}, \tag{2.5}$$

where c is a constant. The only difference between the Pearson correlation and the constrained Pearson correlation is that instead of the mean of the users a value c is substituted. The constrained Pearson correlation can be used to avoid the computation of the mean of the user. As our rating scale is 1 to 5, we assume that the mean vote is 3. If we use $c = 0$, the constrained Pearson correlation is called vector similarity or cosine [3], as the angle between the vectors of users x and y is measured. By choosing a low value for c we can indicate that similarity between high scored songs is more important than similarity between low scored songs.

Example 2.3: If we use the constrained Pearson, with $c = 2$, then the correlation between Aukje and Wim is

$$\frac{3 \cdot 0 + 2 \cdot (-1) - 1 \cdot 2 + 0 \cdot 2 + 0 \cdot 2}{\sqrt{(9+4+1) \cdot (1+4+4+4)}} \approx -0.29$$

The constrained Pearson between Aukje and Arnout is 0.86. The correlation between Arnout and Aukje is stronger compared to the Pearson correlation, while the correlation between Wim and Aukje is less strong.

Counting measures

These measures are based on the number of times that two rated an item similarly. A simple counting measure is the majority voting measure [16], given by

$$s(x,y) = (2-\gamma)^{c_{xy}} \gamma^{w_{xy}}, \tag{2.6}$$

where $0 < \gamma < 1$, and c_{xy} is the number of items that the users rated the same, while w_{xy} is the number of items the users rated differently. More precisely, define a set $C(v)$ as the set of votes that are considered equal to a value v, then $c_{xy} = |\{i \in I_x \cap I_y | v_{yi} \in C(v_{xi})\}|$ and $w_{xy} = |\{i \in I_x \cap I_y | v_{yi} \notin C(v_{xi})\}|$. Usually, the set $C(v)$ is symmetric, i.e. $a \in C(b) \Leftrightarrow b \in C(a)$. In that case we have $c_{xy} = c_{yx}$ and $w_{xy} = w_{yx}$, so the similarity is symmetric too. It can be shown that, if you give the algorithm based on Pearson correlation and the algorithm based on majority voting a

case where they make the most mistakes then majority voting makes fewer mistakes [16]. Here, the algorithm is said to make a mistake if the round off value of the prediction does not equal the actual vote.

In the Jukebox system the kappa statistic [23, 26] was tested as a metric for matching between two users. It was found that it is better to not only look for identical votes, but also to consider nearly-identical votes, although with a lower weight. Therefore a variant was used, called the weighted kappa [14, 15], which is given by

$$s(x,y) = K_{xy}(\omega) = \frac{O_{xy}(\omega) - E_{xy}(\omega)}{1 - E_{xy}(\omega)}. \tag{2.7}$$

Here

$$O_{xy}(\omega) = \sum_{v=1}^{5}\sum_{w=1}^{5} p_{xy}(v,w)\omega_{vw} \tag{2.8}$$

is the observed proportion of agreement and

$$E_{xy}(\omega) = \sum_{v=1}^{5}\sum_{w=1}^{5} q_{xy}(v,w)\omega_{vw} \tag{2.9}$$

is the degree of agreement solely on basis of chance. The probability $p_{xy}(v,w)$ that user x voted v and user y voted w is given by

$$p_{xy}(v,w) = \frac{|\{i \in I_x \cap I_y | v_{xi} = v, v_{yi} = w\}|}{|I_x \cap I_y|} \tag{2.10}$$

If votes were made purely randomly, the estimated expected probability that user x voted v and user y voted w would be $q_{xy}(v,w)$, which is given by

$$q_{xy}(v,w) = \sum_{w'} p_{xy}(v,w') \sum_{v'} p_{xy}(v',w) \tag{2.11}$$

The weights ω_{vw} are chosen such that $0 \leq \omega_{vw} = \omega_{wv} \leq 1 = \omega_{vv}$. These weights reflect the extent in which two votes v and w are considered equal. A good weight matrix is depicted in Table 2.3. Weighted kappa varies from negative values to a value of one indicating perfect agreement.

	1	2	3	4	5
1	1	1/2	0	0	0
2	1/2	1	1/2	0	0
3	0	1/2	1	1/2	0
4	0	0	1/2	1	1/2
5	0	0	0	1/2	1

Table 2.3: A weight matrix for the weighted kappa statistic.

Example 2.4: Here we describe the calculation of the weighted kappa for Aukje and Arnout. First we calculate the fractions p as depicted in Table 2.4. The fractions q are obtained by multiplying the row and column sums of p, for example $q(2,1) = 2/5 \cdot 1/5 = 2/25$. If we use the weight matrix of Table 2.3, then the kappa statistic between Aukje and Arnout is about

$$\frac{0.6 - 0.42}{1 - 0.42} \approx 0.31.$$

The kappa statistic between Aukje and Wim is about -0.56.

	1	2	3	4	5
1	0	1/5	0	0	0
2	0	1/5	1/5	0	0
3	0	0	0	0	0
4	0	0	0	0	1/5
5	0	0	0	1/5	0

Table 2.4: The fraction p between Aukje and Arnout for the example of Table 2.1.

2.2.2 Predictions for the user-based algorithm

In this section we describe three predictors and its variants. We assume that the similarities $s(a, y)$ are chosen such that they are high if the users are similar and negative if they have opposite tastes. We start with the standard predictor given by

$$p_{ai} = \bar{v}_a + \frac{\sum_{y \in U_i} s(a, y)(v_{yi} - \bar{v}_y)}{\sum_{y \in U_i} |s(a, y)|}. \tag{2.12}$$

In Example 2.1 we summed over all the other users. We also could have summed over the users similar to the active user. Often the users are selected according to a certain threshold, and the users with a high similarity (above the threshold) are used to make predictions. A simpler predictor is the following

$$p_{ai} = \frac{\sum_{y \in U_i} s(a, y) v_{yi}}{\sum_{y \in U_i} |s(a, y)|}, \tag{2.13}$$

where the sum is over all users, or over the users with a similarity above a certain threshold. We already mentioned the majority voting similarity measure. The predictor that is used in combination with this similarity is the majority voting predictor, which is given by

$$p_{ai} = \begin{cases} \arg\max_{v \in \{1,\ldots,5\}} \sum_{x : v_{xi} \in C(v)} s(a, x) & \text{if there are values } v_{xi}, \\ c & \text{else,} \end{cases} \tag{2.14}$$

where c is a constant. Note that the majority voting similarity is always positive. The predictor can be adapted in order to deal with negative similarities. To make a prediction, we simply try all the possible votes, the vote with the maximum total similarity is the prediction. With the majority voting predictor, an item can be recommended not only when similar users liked the item, but also when a lot of users liked the item.

Example 2.5: Suppose we define the sets $C(1) = C(2) = \{1, 2\}, C(3) = \{3\}$ and $C(4) = C(5) = \{4, 5\}$. Choose the parameter $\gamma = 0.5$. Now the similarity (2.6) between Aukje and Arnout is equal to

$$s(\text{Aukje,Arnout}) = 1.5^4 \cdot 0.5^1 \approx 2.53.$$

The same can be done for the other pairs of users, resulting in similarity values as shown in Table 2.5. Again we want to calculate a prediction for Aukje and tea T3. The votes 1 and 2 have weight of about $0.03 + 0.03 = 0.06$. Vote 3 has weight 0. Votes 4 and 5 have weight 2.53. The latter votes have the maximum weight, and the prediction is therefore that Aukje likes the tea. As the votes 4 and 5 receive the same weight, the both votes are equally likely.

2.2.3 Online user-based algorithms

The user-based algorithms we described so far, need a total recalculation of the similarities if they become obsolete. Online algorithms adapt the similarities directly as a vote arrives. The similarity for majority voting can be easily translated into an online algorithm. At the beginning of the algorithm

	Wim	Jan	Arnout	Aukje
Wim				
Jan	1.27			
Arnout	0.02	0.02		
Aukje	0.03	0.03	2.53	

Table 2.5: Majority voting similarity for the example of Table 2.1.

the similarity $s(x,y) = 1$, for each pair of users x, y, and we choose a parameter $0 < \gamma < 1$. When a new vote v_{xi} arrives, we set

$$s(x,y) = \begin{cases} (2-\gamma)s(x,y) & \text{if } i \in I_y \text{ and } v_{yi} \in C(v_{xi}) \,, \\ \gamma s(x,y) & \text{if } i \in I_y \text{ and } v_{yi} \notin C(v_{xi}), \\ s(x,y) & \text{if } i \notin I_y. \end{cases} \tag{2.15}$$

The constrained Pearson correlation can be turned into an online algorithm too. Suppose user x and user y have sets I_x and I_y of items they already rated. Now user x makes a new vote v_{xj} for item j. This item is added to the set I_x, creating a new set $I'_x = I_x \cup \{j\}$. Then the new constrained Pearson correlation is given by

$$s(x,y) = \frac{\sum_{i \in I'_x \cap I_y} (v_{xi} - c)(v_{yi} - c)}{\sqrt{\sum_{i \in I'_x \cap I_y} (v_{xi} - c)^2 \sum_{i \in I'_x \cap I_y} (v_{yi} - c)^2}}, \tag{2.16}$$

$$= \frac{\sum_{i \in I_x \cap I_y} (v_{xi} - c)(v_{yi} - c) + (v_{xj} - c)(v_{yj} - c)}{\sqrt{(\sum_{i \in I_x \cap I_y} (v_{xi} - c)^2 + (v_{xj} - c)^2) \cdot (\sum_{i \in I_x \cap I_y} (v_{yi} - c)^2 + (v_{yj} - c)^2)}}, \tag{2.17}$$

provided that item j is rated by user y. If user y did not rate item j, the similarity between the users does not change. To calculate the constrained Pearson correlation incrementally, we maintain three sums for each pair of users, as shown in (2.17). The incremental calculation of the Pearson calculation is a bit more complex, but still possible. The predictions can also be made incremental. The predictions change when the active user a or another user x makes new votes, therefore we distinguish the following four cases.

1. User a makes a new vote for item i, while user x did not rate item i. In this case the similarity $s(a,x)$ does not change, therefore the predictions do not change. Of course, the prediction for user a and item i is not necessary anymore.

2. User x makes a new vote for item i, while user a did not rate item i. In this case the similarity $s(a,x)$ does not change, however the set U_i is changed to $U'_i = U_i \cup \{x\}$. Hence, the prediction for item i is changed into

$$p_{ai} = \frac{\sum_{y \in U'_i} s(a,y) v_{yi}}{\sum_{y \in U'_i} |s(a,y)|} = \frac{\sum_{y \in U_i} s(a,y) v_{yi} + s(a,x) v_{xi}}{\sum_{y \in U_i} |s(a,y)| + |s(a,x)|},$$

   where we use the simple predictor, which is given by (2.13).

3. User x makes a new vote for item i, while user a already rated item i. Now the similarity $s(a,x)$ changes into $s'(a,x) = s(a,x) + \Delta$. The predictions with the simple predictor (2.13) for the items $j \in I_x$ that user a did not rate, change as follows

$$p_{aj} = \frac{\sum_{y \in U_i \setminus \{x\}} s(a,y) v_{yj} + s'(a,x) v_{xj}}{\sum_{y \in U_i \setminus \{x\}} |s(a,y)| + |s'(a,x)|} = \frac{\sum_{y \in U_i \setminus \{x\}} s(a,y) v_{yj} + (s(a,x) + \Delta) v_{xj}}{\sum_{y \in U_i \setminus \{x\}} |s(a,y)| + |s(a,x) + \Delta|}$$

$$= \frac{\sum_{y \in U_i} s(a,y) v_{yj} + \Delta v_{xj}}{\sum_{y \in U_i} |s(a,y)| - |s(a,x)| + |s'(a,x)|}.$$

4. User a makes a new vote for item i, while user x already rated item i. The predictions can be adapted in the way described in the preceding case. Of course, the prediction for item i is not necessary anymore.

The simple predictor can thus be made incremental by maintaining the denominator and the nominator separately. The standard predictor and the majority voting predictor can be made incremental in a similar way.

2.3 Item-based algorithms

Item-based algorithms [13, 21] act oppositely to user-based algorithms in the sense that instead of calculating similarities between users, similarities between items are calculated. The adjusted cosine similarity

$$s(i,j) = \frac{\sum_{x \in U_i \cap U_j} (v_{xi} - \bar{v}_x)(v_{xj} - \bar{v}_x)}{\sqrt{\sum_{x \in U_i \cap U_j} (v_{xi} - \bar{v}_x)^2} \sqrt{\sum_{x \in U_i \cap U_j} (v_{xj} - \bar{v}_x)^2}} \tag{2.18}$$

can be used to calculate matchings between items. The difference between the Pearson correlation and the adjusted cosine is that the mean of the user is subtracted instead of the mean of the item. In [21] the correlation and the adjusted cosine were tested with an item-based algorithm. The latter was found the best. Some users tend to give higher votes than other users. If we want to compare the votes of two items for different users, we have to scale it according to the user. Otherwise, users who tend to give high votes have too much influence. The standard item-based predictor for user a and item j is given by

$$p_{aj} = \bar{v}_j + \frac{\sum_{i \in I_a} s(j,i)(v_{ai} - \bar{v}_i)}{\sum_{i \in I_a} |s(j,i)|}. \tag{2.19}$$

Now we need to calculate $O(I^2)$ similarities, while the sums are calculated over the users. Therefore the similarity-calculation phase costs $O(I^2U)$ time. The prediction phase has the same time complexity. The total algorithm has a time complexity of $O(I^2U)$.

The similarity measures and predictors for the user-based algorithm can all be turned into similarity measures and predictors for the item-based algorithms. We expect the item-based algorithm to have about the same performance as the user-based algorithm. The item-based algorithm should theoretically be used if there are more users than items, then the algorithm is faster, but also the predictions will be better, as the similarities are based on more data.

2.4 Factor analysis

Factor analysis is a generalization of singular value decomposition [5, 7, 20, 2] and linear regression [4]. With factor analysis we try to make a linear model that can describe the users preferences, given by

$$V = \Lambda X + N,$$

where V is an $I \times U$ matrix with the user profiles as columns. We assume that the user profiles are generated by a random process, which depends on X. The entries of the $k \times U$ matrix X are assumed to be standard normally distributed. The noise (N) is normally distributed with mean 0 and variance ψ. Hence, the user profiles are normally distributed with mean 0 too. In order to satisfy this assumption, we subtract the mean of the users from their profiles. The $I \times U$ matrix N represents the error that is made when we approximate V by ΛX. The $I \times k$ matrix Λ consists of k basis vectors. The columns of the matrix X give the combinations of these vectors needed to approximate each user in V. The matrix Λ is calculated in such a way that the noise N is minimized. To build the model we use an EM algorithm, where as initialization we take a random model Λ and $\psi = 1$. Then the algorithm proceeds

with an iteration:

$$\begin{aligned} M &= (\psi I + \Lambda^T \Lambda)^{-1}, \\ X &= M\Lambda^T V, \\ \Lambda &= VX^T(XX^T + U\psi M)^{-1}, \\ \psi &= \frac{1}{UI}\mathrm{tr}(VV^T - \Lambda X V^T), \end{aligned} \tag{2.20}$$

where tr (trace) is the sum of the diagonal elements of a matrix. More about the EM algorithm and its application to factor analysis can be found in [8, 22, 4]. The time complexity of this algorithm is $O(UIk^2)$. The number of basis vectors k is usually chosen small (less than twenty). In order to secure the user profiles we split the calculations (2.20) in Section 2.4.1 between the users and the server.

Example 2.6: If we run a factor-analysis algorithm with $k = 2$ on the tea and coffee example we get the model depicted in Figure 2.2. Before the iteration starts, we subtract the mean votes, such that the mean of the user profiles is 0.

$$\begin{pmatrix} 2.2 \\ 1.2 \\ ? \\ -1.8 \\ 0.2 \\ -0.8 \\ ? \\ -0.8 \\ ? \end{pmatrix} = \begin{pmatrix} -0.45 & 1 \\ -0.99 & 0.07 \\ -1 & 1 \\ 0.8 & -0.72 \\ -1 & -0.69 \\ 0.49 & -0.29 \\ 0.21 & -0.31 \\ 0.81 & 0.08 \\ 0.15 & -0.09 \end{pmatrix} \cdot \begin{pmatrix} -1.1 \\ 1.5 \end{pmatrix} + \begin{pmatrix} 0.21 \\ 0.01 \\ ? \\ 0.16 \\ 0.14 \\ 0.17 \\ ? \\ -0.03 \\ ? \end{pmatrix}$$

Figure 2.2: $\mathbf{v}_{\text{Aukje}} = \Lambda \mathbf{x}_{\text{Aukje}} + \mathbf{n}_{\text{Aukje}}$ for the example of Table 2.1.

The matrix Λ has two columns corresponding to the two basis vectors. Aukje has a combination for these two basis vectors of $\mathbf{x}_{\text{Aukje}} = \{-1.1, 1.5\}$. The prediction for flavour T3 is calculated by multiplying row T3 of the matrix Λ and the vector $\mathbf{x}$ of Aukje and adding her mean vote, $p_{\text{Aukje,T3}} = -1 \cdot (-1.1) + 1 \cdot 1.5 + 2.8 = 5.4$. We can conclude that Aukje is incredibly fond of tea number three. We round the prediction down to 5, as this is our maximum vote.

2.4.1 Recurrence relation

The EM algorithm for factor analysis consists of two steps. In the first step (combination calculation) we calculate the combinations X such that they describe the user profiles as good as possible given the model Λ. In the second step (model calculation) we calculate a new model Λ based on these combinations X such that the noise is minimized.

Combination calculation. Given model Λ and variance ψ we calculate a combination X that describes the matrix V the best.

$$\begin{aligned} M &= (\psi I + \Lambda^T \Lambda)^{-1} \\ X &= M\Lambda^T V \end{aligned}$$

These calculations can be split among the users as follows

$$\begin{aligned} \Lambda_y &= R_y \Lambda \\ M_y &= (\psi I + \Lambda_y^T \Lambda_y)^{-1} \\ \mathbf{x}_y &= M_y \Lambda_y^T \mathbf{v}_y \end{aligned} \tag{2.21}$$

The matrix R_y is a diagonal matrix with elements $(R_y)_{ii} = r_{yi}$. The rows of the matrix Λ correspond to the items in the system. Row i of matrix Λ_y is 0 when item i is not rated by user y. If item i is rated by user y, row i of matrix Λ_y is equal to row i of Λ. Note that the items the user did not rate are not taken into account in the calculation of the combination $\mathbf{x}_y$.

Model calculation. Given the database V and the combinations X, we calculate a new model Λ with minimum variance ψ.

$$\begin{aligned} \Lambda &= VX^T(XX^T + U\psi M)^{-1} \\ \psi &= \frac{1}{UI}\mathrm{tr}(VV^T - \Lambda XV^T) \end{aligned}$$

The first equation can also be written as $\Lambda(XX^T + U\psi M) = VX^T$. Splitting the calculations over the users gives us

$$\sum_{y=1}^{U} R_y \Lambda(\mathbf{x}_y\mathbf{x}_y^T + \psi M_y) = \sum_{y=1}^{U} \mathbf{v}_y\mathbf{x}_y^T.$$

Now we make a vector of the matrix Λ by putting the rows of the matrix behind each other, and denote this vector by $L(\Lambda)$. Similarly we have $L(\sum_{y=1}^{U} \mathbf{v}_y\mathbf{x}_y^T)$. Furthermore, we want to use the kronecker product $\otimes$.

Definition 2.1 *Let A be an $n_A \times m_A$ matrix with elements a_{ij} and let B be an $n_B \times m_B$ matrix, then*

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1m_A}B \\ \vdots & \ddots & \vdots \\ a_{n_A1}B & \dots & a_{n_Am_A}B \end{pmatrix}.$$

This matrix consists of $n_A \times m_A$ blocks of size $n_B \times m_B$.

Then we have the equality

$$\sum_{y=1}^{U} R_y \otimes (\mathbf{x}_y\mathbf{x}_y^T + \psi M_y)^T L(\Lambda) = L(\sum_{y=1}^{U} \mathbf{v}_y\mathbf{x}_y^T).$$

Now we only have to split the calculation for ψ between the users. As the trace is the sum of the diagonal elements of a matrix, we have that $\mathrm{tr}(VV^T - \Lambda XV^T) = \mathrm{tr}(VV^T) - \mathrm{tr}(\Lambda XV^T)$. The first term can be rewritten as

$$\mathrm{tr}(VV^T) = \sum_{i=1}^{I}\sum_{y=1}^{U} v_{iy}v_{yi}^T = \sum_{y=1}^{U}\sum_{i=1}^{I} v_{iy}v_{yi}^T = \sum_{y=1}^{U} \mathbf{v}_y^T\mathbf{v}_y.$$

The second term is obtained in the same way,

$$\mathrm{tr}(\Lambda XV^T) = \sum_{i=1}^{I}\sum_{j=1}^{k}\Lambda_{ij}\sum_{y=1}^{U} x_{jy}v_{yi} = \sum_{y=1}^{U}\sum_{i=1}^{I}\sum_{j=1}^{k}\Lambda_{ij}x_{jy}v_{yi} = \sum_{y=1}^{U}\mathrm{tr}(\Lambda\mathbf{x}_y\mathbf{v}_y^T).$$

Now we have

$$\psi = \frac{1}{\sum_{y=1}^{U}|I_y|}\sum_{y=1}^{U}(\mathbf{v}_y^T\mathbf{v}_y - \mathrm{tr}(\Lambda\mathbf{x}_y\mathbf{v}_y^T)).$$

The total number of votes is equal to $\sum_{y=1}^{U}|I_y|$. In the algorithm, the users can calculate the parts

$$\begin{aligned} a_y &= (\mathbf{x}_y\mathbf{x}_y^T + \psi M_y)^T, \\ A_y &= R_y \otimes a_y, \\ B_y &= \mathbf{v}_y\mathbf{x}_y^T, \\ C_y &= \mathbf{v}_y^T\mathbf{v}_y. \end{aligned} \tag{2.22}$$

The server calculates the sums over the users and makes a new model

$$\begin{aligned} L(\Lambda) &= (\sum_{y=1}^{U} A_y)^{-1} L(\sum_{y=1}^{U} B_y), \\ \psi &= \frac{1}{\tilde{I}} \sum_{y=1}^{U} C_y - L(\Lambda)^T L(\sum_{y=1}^{U} B_y). \end{aligned} \tag{2.23}$$

In the calculation of ψ we use that

$$\sum_{y=1}^{U} \mathrm{tr}(\Lambda \mathbf{x}_y \mathbf{v}_y^T) = \mathrm{tr}(\Lambda \sum_{y=1}^{U} B_y^T) = L(\Lambda)^T L(\sum_{y=1}^{U} B_y).$$

The value $\tilde{I}$ is an approximation for $\sum_{y=1}^{U} |I_y|$, the number of votes. In the first iteration we can for instance make an approximation for $\sum_{y=1}^{U} |I_y|$ by computing $\tilde{I}$ such that $\psi = 1$.

When a user wants recommendations, he first downloads Λ and ψ. With this model he calculates his combination $\mathbf{x}$ via (2.21). By multiplying he can calculate the predictions Λx.

Chapter 3

Encryption

This chapter describes the building blocks that we use to achieve the desired security. Most of the methods for encryption rely on a problem that is easy to solve when some knowledge (a secret key) is available and difficult to solve when that specific knowledge is not available.

3.1 Basic operations

The user-based algorithm requires the calculation of inner products between profiles of different users. The item-based algorithm requires the calculation of sums over the users, where the elements have to remain secret.

Example 3.1: Aukje rated tea T1 and T2, Jan rated tea T1 and T3. If they want to know how many teas they both rated without giving information about which teas they exactly rated, they need a protocol to calculate the inner product between their rating vectors. The rating vector $\mathbf{r}_A$ of Aukje is $(1,1,0)$ and the vector $\mathbf{r}_J$ of Jan is $(1,0,1)$. The inner product between these vectors is 1, which is precisely the number of teas they both rated. Of course, Aukje has to protect her data, which she does by applying an encryption function $\varepsilon(r)$, where r is an element of her rating vector $\mathbf{r}_A$. In this way, she obtains the encrypted vector $(\varepsilon(r_{A,T1}), \varepsilon(r_{A,T2}), \varepsilon(r_{A,T3}))$. Because nobody is allowed to read her data, we need an operation $MULTIPLY$ with the property

$$MULTIPLY(m_1, \varepsilon(m_2)) = \varepsilon(m_1 \cdot m_2), \tag{3.1}$$

and an operation SUM with the property

$$SUM(m_1, m_2) = \varepsilon(m_1 + m_2). \tag{3.2}$$

Jan elementswise multiplicates his vector with the encrypted vector of Aukje, by applying the $MULTIPLY$ protocol. This results in the vector $(\varepsilon(r_{J,T1}r_{A,T1}), \ldots, \varepsilon(r_{J,T3}r_{A,T3}))$. Finally, he applies the SUM protocol to obtain the encrypted inner product, while the data of Aukje is not revealed to him. The inner product is decrypted by Aukje, as she has the secret key. A cryptosystem which possesses the desired properties for the SUM and $MULTIPLY$ operations is described in Section 3.2.

Example 3.2: Aukje, Jan, Wim and Arnout want to know how many times an opinion is given about tea T2, without revealing whether they rated the tea or not. T2 is rated by Wim, Arnout and Aukje and not rated by Jan, i.e. $r_{W,T2} = 1, r_{Ar,T2} = 1, r_{Au,T2} = 1$, and $r_{J,T2} = 0$. So there are three persons who gave an opinion about tea T2. To protect these date, they apply an encryption function $\varepsilon(r)$. The encrypted sum $\varepsilon(r_{W,T2} + r_{Ar,T2} + r_{Au,T2} + r_{J,T2})$ is obtained via a SUM operation as described in the previous example. The problem is that none of them can have the secret key. The solution is to share the key among the users. They can only decrypt a message if they cooperate. A cryptosystem

with key sharing is described in Section 3.3. Another solution is to use two independent servers. The first server calculates the encrypted sum, the second server has the key and can decrypt the message.

A system that has the properties described in the above examples is the so-called Paillier system, which is described in the next section. We conclude this section with some notations that we will use in the remaining sections.

- Let $\mathbb{Z}_n^* = \{x \in \mathbb{Z} | 0 < x < n, \gcd(x,n) = 1\}$ denote the multiplicative subgroup of integers modulo n. The size of this group is denoted by $\varphi(n)$. If the prime decomposition of $n = p_1^{e_1} p_2^{e_2} ... p_k^{e_k}$ then $\varphi(n) = p_1^{e_1-1}(p_1-1)p_2^{e_2-1}(p_2-1)...p_k^{e_k-1}(p_k-1)$.
- The Carmichael's function $\lambda(n)$ is defined as the smallest integer m such that for all $w \in \mathbb{Z}_n^*$ $w^m \bmod n = 1$. We can calculate the values for $\lambda(n)$ with the following recurrence relation.

$$\lambda(n) = \begin{cases} \varphi(n) & \text{if } n = p^e \text{ and } (p = 2 \wedge e \leq 2) \vee (p \geq 3), \\ \frac{1}{2}\varphi(n) & \text{if } n = 2^e \text{ and } e \geq 3, \\ \operatorname{lcm}(\lambda(2^e), \lambda(p_1^{e_1}), \ldots, \lambda(p_k^{e_k})) & \text{if } n = 2^e \prod_{i=1}^{k} p_i^{e_i}. \end{cases} \tag{3.3}$$

  In accordance with the definition of $\lambda(n)$, Carmichael's theorem states that if $w \in \mathbb{Z}_n^*$ then $w^{\lambda(n)} \equiv 1 \pmod{n}$.
- An integer z is said to be an r-th residue modulo n if and only if there exists some integer $x \in \mathbb{Z}_n^*$ such that $z = x^r \bmod n$.
- For each positive integer n, the size (or order) of n is defined by $|n| = \lceil \log_2 n \rceil$.

3.2 The Paillier cryptosystem

This section introduces the Paillier cryptosystem [17]. This system possesses the desired properties of the operations $MULTIPLY$ and SUM as defined in Example 3.1. The implementation of these operations is described in Section 3.2.2, which describes the algorithms needed for encryption and decryption too. The subsequent section describes some theory behind these algorithms.

3.2.1 Basics

First we explain the problem where the cryptosystem of Paillier relies on, which is called the composite degree residuosity class problem. Next, we show how the problem can be solved when we have some specific knowledge. Before the problem is explained, we define set $\mathcal{B}$ as

$$\mathcal{B} = \{g \in \mathbb{Z}_{n^2}^* | |g| = |\alpha n| \text{ for an } \alpha = 1, \ldots, \lambda(n)\}.$$

Definition 3.1 (Composite degree residuosity class problem) *Given $w \in \mathbb{Z}_{n^2}^*$ and $g \in \mathcal{B}$ try to find the residuosity class $x \in \mathbb{Z}_n$ for which there exists a $y \in \mathbb{Z}_n^*$ such that*

$$g^x y^n \bmod n^2 = w,$$

The class of w with respect to g is denoted by $[w]_g$.

If $g \in \mathcal{B}$ we have that the function $\varepsilon_g : \mathbb{Z}_n \times \mathbb{Z}_n^* \longmapsto \mathbb{Z}_{n^2}^*$ defined by

$$\varepsilon_g(x,y) = g^x y^n \bmod n^2$$

is bijective, i.e. the composite residuosity class problem has a unique solution given a $w \in \mathbb{Z}_{n^2}^*$. If we choose $n = pq$, where p and q are two primes, then the Carmichael's function $\lambda(n) = \operatorname{lcm}(p-1, q-1)$. In the following, we will refer to this $\lambda(n)$ as λ. Due to Carmichael's theorem we have that for any $w \in \mathbb{Z}_{n^2}^*$

$$w^{\lambda} \equiv 1 \pmod{n} \tag{3.4}$$

$$w^{n\lambda} \equiv 1 \pmod{n^2} \tag{3.5}$$

Theorem 3.1 *For any* $w \in \mathbb{Z}_{n^2}^*$, $w^\lambda \equiv 1 + [w]_{1+n}\lambda n \pmod{n^2}$.

Proof: As $|1+n| = |n|$, we have that $(1+n) \in \mathcal{B}$, so there exists a unique pair $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_n^*$ with the property that $w = (1+n)^a b^n \bmod n^2$. By definition, $a = [w]_{1+n}$. By making use of (3.5) we have

$$w^\lambda = (1+n)^{a\lambda} b^{n\lambda} = (1+n)^{a\lambda}.$$

Because $(1+n)^x \equiv 1 + xn \pmod{n^2}$ we have that

$$(1+n)^{a\lambda} \equiv 1 + a\lambda n \pmod{n^2},$$

which yields the result.

Definition 3.2 *For any* $u \in \{x < n^2 | x \equiv 1 \pmod{n}\}$, *we define*

$$L(u) = \frac{u-1}{n}.$$

By making use of Theorem 3.1 the residuosity class $[w]_g$ is

$$\frac{L(w^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n = \frac{\lambda[w]_{1+n}}{\lambda[g]_{1+n}} = [w]_g,$$

where the division is calculated by multiplying with the inverse of $L(g^\lambda \bmod n^2)$ modulo n. The latter equality can be shown by using the fact that by definition $g = (n+1)^{[g]_{n+1}} b_1^n \bmod n^2$ for a certain $b_1 \in \mathbb{Z}_n^*$ and hence,

$$g^{\frac{[w]_{n+1}}{[g]_{n+1}}} b_2^n \bmod n^2 = (n+1)^{[w]_{n+1}} b_3^n \bmod n^2 = w,$$

where $b_2 \in \mathbb{Z}_n^*$ and $b_3 = b_2^{[w]_g} b_1 \in \mathbb{Z}_n^*$.

3.2.2 Encryption and decryption

In short, the encryption and decryption of a message goes as follows

Key generation. Choose two large primes p, q and $g \in \mathcal{B}$ and calculate $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. The public key is the pair (n, g), the private key is λ.

Encryption. The user who wants to send a message $m \in \mathbb{Z}_n$ to a receiver with public keys n and g makes a ciphertext $c = \varepsilon(m) = g^m r^n \bmod n^2$ where r is randomly chosen from $\mathbb{Z}_n^*$. In this way the message m can not be obtained by trying the possible values of m.

Decryption. The receiver can obtain the message m via:

$$m = \delta(c) = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n,$$

where the division is taken by multiplying with the inverse of $L(g^\lambda \bmod n^2)$ modulo n. In Example 3.1 we used a SUM and $MULTIPLY$ function. In the Paillier system, the SUM function is implemented as a multiplication of the ciphertexts, and the $MULTIPLY$ function is implemented by simply taking powers, as

$$\varepsilon(m_1)\varepsilon(m_2) = g^{m_1} r_1^n g^{m_2} r_2^n \bmod n^2 = g^{m_1+m_2}(r_1 r_2)^n \bmod n^2 = \varepsilon(m_1 + m_2), \quad (3.6)$$

$$\varepsilon(m_1)^{m_2} = (g^{m_1} r^n)^{m_2} \bmod n^2 = g^{m_1 m_2}(r^{m_2})^n \bmod n^2 = \varepsilon(m_1 m_2). \quad (3.7)$$

A ciphertext can always be changed into another ciphertext without affecting the plaintext with the property given by

$$\varepsilon(m_1) r^n = g^{m_1} r_1^n r^n \bmod n^2 = g^{m_1}(r_1 r)^n \bmod n^2 = \varepsilon(m_1), \quad (3.8)$$

where the ciphertext of the message m_1 is changed by multiplying with r^n where r is randomly chosen from $\mathbb{Z}_n^*$.

Example 3.3: Choose $p = 5$ and $q = 7$, then $n = 35, n^2 = 1225$, and $\lambda = 12$. A valid choice for $g = 36$, as $|36| = 6$ and $|1 \cdot 35| = 6$. Now Aukje can encrypt her vector $(1, 1, 0)$ as follows

$$\begin{aligned} 36^1 \cdot 3^{35} \bmod 1225 &= 1027 \\ 36^1 \cdot 4^{35} \bmod 1225 &= 639 \\ 36^0 \cdot 29^{35} \bmod 1225 &= 99 \end{aligned}$$

She sends her encrypted vector to Jan. Jan applies the $MULTIPLY$ and SUM operations $1027^1 \cdot 639^0 \cdot 99^1 \bmod 1225 = 1223$ and sends the encrypted inner product back to Aukje. Aukje knows the secret key, so she can calculate the inner product by applying the decryption function

$$\frac{((1223^{12} \bmod 1225) - 1)/35}{((36^{12} \bmod 1225) - 1)/35} \bmod 35 = 12 \cdot 3 \bmod 35 = 1,$$

where 3 is the inverse of 12 modulo 35.

3.3 The threshold version of the Paillier cryptosystem

A threshold cryptosystem [9] allows any subset of $t + 1$ out of ℓ users to decrypt the ciphertext but prevents decryption if at most t users participate.

Key generation.

- Choose two strong primes p, q, i.e. $p = 2p' + 1, q = 2q' + 1$, where p' and q' are prime too. Set $n = pq$ and $n' = p'q'$.
- Choose random $\beta, a, b \in \mathbb{Z}_n^*$ and set $g = (1 + n)^a b^n \bmod n^2$. The secret key $s = \beta n'$. Set $\theta = an'\beta \bmod n$.
- The public key is the triple (n, g, θ) and the private key is s which is shared among the users with key sharing.

Key sharing. The Shamir key sharing scheme [24] is based on polynomial interpolation. We can retrieve a polynomial $f(X) = \sum_{i=0}^{t} a_i X^i$ of degree t with unknown coefficients a_i, if the values $f(x_i)$ of $t + 1$ points x_i are known, i.e.

$$f(X) = \sum_{i=1}^{t+1} \prod_{j=1, j\neq i}^{t+1} \frac{X - x_j}{x_i - x_j} f(x_i).$$

To share the key s among the users, we choose a polynomial of degree t such that $f(0) = a_0 = s$. The other coefficients a_i, where $0 < i \leq t$ are randomly chosen from $\mathbb{Z}_{nn'}$. Now we give each user u a point (u, s_u), where $s_u = f(u) \bmod nn'$. We call s_u the share of user u. Then the secret key s can be reconstructed via:

$$s = f(0) = \sum_{u\in\Omega} \prod_{y\in\Omega\setminus\{u\}} \frac{-y}{u - y} f(u) = \sum_{u\in\Omega} s_u L_{u\Omega},$$

where Ω is a set of $t + 1$ users with a share s_u, and $L_{u\Omega} = \prod_{y\in\Omega\setminus\{u\}} \frac{y}{y-u}$.

Encryption. To encrypt a message $m \in \mathbb{Z}_n$, compute $c = \varepsilon(m) = g^m r^n \bmod n^2$ with a random $r \in \mathbb{Z}_n^*$ just as in the normal Paillier system.

Share decryption. Each user u computes a decryption share $c_u = \delta'(c) = c^{2\Delta s_u} \bmod n^2$, where $\Delta = \ell!$. Note that the share s_u of user u, is hidden in the same way as the message m.

End decryption. Let Ω be a set of $t+1$ valid shares. Then the message m is retrieved by:

$$m = \delta_T(C) = L(\prod_{u\in\Omega} c_u^{2\mu_u} \bmod n^2)\frac{1}{4\Delta^2\theta} \bmod n$$

where $\mu_u = \Delta L_{u\Omega}$ and $C = \{c_u | u \in \Omega\}$.

Example 3.4: Choose $p' = 2$ and $q' = 3$ then $p = 5, q = 7, n = 35, n^2 = 1225$, and $n' = 6$. Choose $g = 36$ (i.e. $a = 1$ and $b = 1$), and $\beta = 4$, resulting in $s = 6 \cdot 4 = 24$ and $\theta = 24$. There are four users, so $\Delta = 4! = 24$. The public keys are $\theta = 24, n = 35, g = 36$. We share the secret key $s = 24$ in such a way that two users must collaborate to decrypt the message (i.e. $t = 1$). We hence make a function $f(x) = 24 + 3x$ with degree 1, where the first coefficient is s and the second coefficient (3) is randomly chosen. Wim gets the share $24 + 3 \cdot 1 = 27$ and Jan,Arnout,Aukje get the shares $f(2) = 30$, $f(3) = 33$, and $f(4) = 36$, respectively. With the use of the public keys and the SUM operation they can encrypt their messages

Wim:	$36^1 3^{35} \bmod 1225$	=	1027,
Jan:	$36^0 29^{35} \bmod 1225$	=	99,
Arnout:	$36^1 4^{35} \bmod 1225$	=	639,
Aukje:	$36^1 2^{35} \bmod 1225$	=	648,
SUM:	$1027 \cdot 99 \cdot 639 \cdot 648 \bmod 1225$	=	1181.

Now we need two users who calculate the decrypted share

Wim:	$1181^{2\cdot 24\cdot 27} \bmod 1225$	=	106,
Jan:	$1181^{2\cdot 24\cdot 30} \bmod 1225$	=	526.

The end decryption calculates the message

$$((106^{2\cdot 48} \cdot 526^{2\cdot -24} \bmod 1225) \cdot (4 \cdot 24^2 \cdot 24)^{-1}) \bmod 35 = (23 \cdot 26) \bmod 35 = 3,$$

where $\Omega = \{1, 2\}$, $\mu_1 = \Delta\frac{2}{2-1} = 48$ and $\mu_2 = \Delta\frac{1}{1-2} = -24$. The number of users that rated T2 is 3.

3.4 The El Gamal cryptosystem

Every cryptosystem that has an implementation for the SUM and $MULTIPLY$ operations can in principle be used instead of the Paillier system. Sometimes a little trick can be applied to obtain the desired properties, such as in the El Gamal cryptosystem [10]. This system works as follows.

Key generation. Choose two primes p, q such that $q|p-1$, and a value $g \in \mathbb{Z}_q^*$. Choose a secret key $s \in \mathbb{Z}_q$. The public keys are p, q, g and $g^s \bmod p$.

Encryption. Choose a random $r \in \mathbb{Z}_q$. Then the encryption function is

$$(c_1, c_2) = \varepsilon(m) = (g^r \bmod p, mg^{sr} \bmod p),$$

where $g^{sr} \bmod p$ is calculated by raising $g^s \bmod p$ to the power r.

Decryption. The decryption consists of two steps,

$$\begin{aligned} \delta_1(c_1) &= c_1^s \bmod p = g^{sr} \bmod p, \\ \delta_2(c_2) &= c_2\delta_1^{-1}(c_1) \bmod p = mg^{sr}g^{-sr} \bmod p = m, \end{aligned}$$

where g^{-sr} is the inverse of g^{sr} in $\mathbb{Z}_p$. The system has the property that

$$\varepsilon(m_1)\varepsilon(m_2) = \varepsilon(m_1 m_2), \tag{3.9}$$

as

$$\begin{aligned} g^{r_1} g^{r_2} \bmod p &= g^{r_1+r_2} \bmod p, \\ m_1 g^{r_1} m_2 g^{r_2} \bmod p &= m_1 m_2 g^{r_1+r_2} \bmod p. \end{aligned}$$

We can turn the El Gamal cryptosystem into a system with SUM and $MULTIPLY$ operations by encrypting γ^m instead of m, where $\gamma \in \mathbb{Z}_q^*$ and γ is known to all users in the system. The message m can be retrieved by trying all the possible messages.

Example 3.5: Suppose we choose $p = 11, q = 5, g = 3, s = 4$, and $\gamma = 2$. Then Aukje encrypts her vector as follows:

$$\begin{aligned} \varepsilon(\gamma^1) &= (3^1 \bmod 11, 2^1 3^{4\cdot 1} \bmod 11) = (3, 8) \\ \varepsilon(\gamma^1) &= (3^2 \bmod 11, 2^1 3^{4\cdot 2} \bmod 11) = (9, 10) \\ \varepsilon(\gamma^0) &= (3^3 \bmod 11, 2^0 3^{4\cdot 3} \bmod 11) = (5, 9) \end{aligned}$$

Jan applies the $MULTIPLY$ and SUM operations of the El Gamal system, $(3^1 \cdot 9^0 \cdot 5^1 \bmod 11, 8^1 \cdot 10^0 \cdot 9^1 \bmod 11) = (4, 6)$. He sends his result back to Aukje. Aukje calculates $6 \cdot 4^{-4} \bmod 11 = 6 \cdot 4 \bmod 11 = 2$. The inproduct is 1, as $\gamma^1 = 2$.

In the same way other systems with property (3.9) can be turned into systems with the same properties as the Paillier system. A disadvantage of these systems is the search for the correct message in the last step, which can be time consuming. Therefore the system is only used when m takes a limited number of values. The El Gamal system has a threshold version too, which is obtained in the same way as the Paillier threshold system. A more detailed description of the El Gamal threshold system is given in [5, 6].

Chapter 4

Protocols for the User-Based Algorithm

In this chapter we shall derive protocols needed for the protection of valuable data in user-based algorithms.

4.1 Protocols for the similarities

In this section we derive protocols for the similarity measures mentioned in Section 2.2.1.

4.1.1 Protocols for the distances

Mean-square difference. The mean-square difference can be rewritten as

$$\frac{\sum_{i\in I_a\cap I_x}(v_{ai}-v_{xi})^2}{|I_a\cap I_x|}=\frac{\sum_{i\in I_a\cap I_x}(v_{ai}^2-2v_{ai}v_{xi}+v_{xi}^2)}{|I_a\cap I_x|}=\frac{\mathbf{r}_x'\mathbf{v}_a^2-2\mathbf{v}_a'\mathbf{v}_x+\mathbf{r}_a'\mathbf{v}_x^2}{\mathbf{r}_a'\mathbf{r}_x},$$

where the vector $\mathbf{v}_x^2$ is the vector with elements v_{xi}^2, and items that are not rated receive a zero vote, i.e. $v_{xi}=0$. The mean-square difference consists of four inner products between the users. These four inner products can be calculated in the way described in Section 3.2.2. The active user calculates his vectors $\mathbf{r}_a$, $\mathbf{v}_a$ and $\mathbf{v}_a^2$ first, as shown in Figure 4.1. He encrypts all entries in the vectors with the encryption function of the Paillier system. These vectors are sent to the server. The server sends the vectors to the other users in the system. The other users have already calculated their vectors $\mathbf{r}_x$, $\mathbf{v}_x$ and $\mathbf{v}_x^2$. They can calculate the four encrypted inner products, $\varepsilon(\mathbf{r}_a'\mathbf{r}_x)$, $\varepsilon(\mathbf{r}_a'\mathbf{v}_x^2)$, $\varepsilon(\mathbf{v}_a'\mathbf{v}_x)$, and $\varepsilon(\mathbf{v}_a^{2'}\mathbf{r}_x)$ by taking powers and multiplying all elements in a vector. These inner products are sent to the server and the server sends it to the active user. The active user can decrypt the four inner products and calculate the mean-square difference. The sum $\mathbf{r}_x'\mathbf{v}_a^2$ can reveal information about items another user rated. As the server takes care of the conversation between two users, the other user stays anonymous and the active user only knows the value of the correlation with another (anonymous) user.

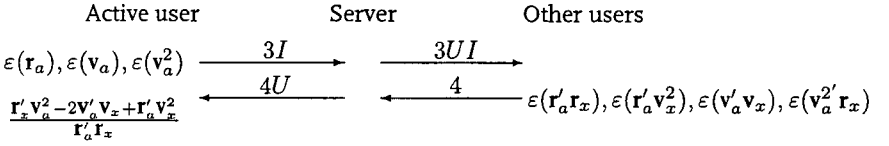

Figure 4.1: Protocol for the mean-square difference. The numbers above the arrows denote the number of messages that are sent by the active user a, by the server or by another user x.

Manhattan distance. We were not able to calculate absolute values in a secured way. We can calculate the encrypted differences in votes between two users, but then we have to decrypt the difference in order to see if it is positive or negative.

4.1.2 Protocol for the correlations

Pearson correlation. We start with rewriting the Pearson correlation. Let us define a vector $\mathbf{w}_x$ with elements

$$w_{xi} = \begin{cases} v_{xi} - \bar{v}_x & \text{if } i \in I_x, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

We write $\mathbf{w}^2$ for the vector with elements w_{xi}^2. Then the vector notation of the Pearson correlation is

$$s(a,x) = \frac{\sum_{i \in I_a \cap I_x} (v_{ai} - \bar{v}_a)(v_{xi} - \bar{v}_x)}{\sqrt{\sum_{i \in I_a \cap I_x} (v_{ai} - \bar{v}_a)^2 \sum_{i \in I_a \cap I_x} (v_{xi} - \bar{v}_x)^2}} = \frac{\mathbf{w}_a' \mathbf{w}_x}{\sqrt{\mathbf{r}_x' \mathbf{w}_a^2 \mathbf{r}_a' \mathbf{w}_x^2}}. \tag{4.2}$$

The correlation consists of three inner products between the users. We can use the Paillier system as explained in Section 3.2.2 to calculate this inner products without giving information about the users tastes.

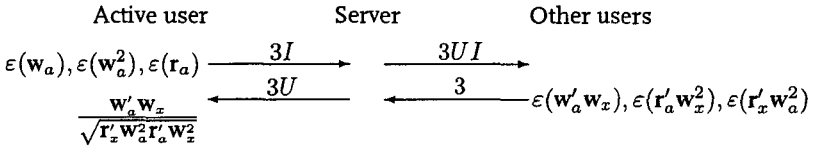

Figure 4.2: Protocol for the Pearson correlation.

The active user sends the encrypted vectors $\varepsilon(\mathbf{w}_a)$, $\varepsilon(\mathbf{w}_a^2)$ and $\varepsilon(\mathbf{r}_a)$ to the other users, as shown in Figure 4.2. The other users calculate the three encrypted inner products, $\varepsilon(\mathbf{w}_a'\mathbf{w}_x)$, $\varepsilon(\mathbf{r}_a'\mathbf{w}_x^2)$ and $\varepsilon(\mathbf{r}_x'\mathbf{w}_a^2)$. These inner products are sent via the server to the active user. The active user can decrypt the three inner products and calculate the Pearson correlation. The server should take care that the other users stay anonymous. The inner product $\mathbf{r}_x'\mathbf{w}_a^2$ in the nominator reveals some information about items another user rated.

Constrained Pearson correlation. The constrained Pearson correlation can be secured in an identical way as the Pearson correlation.

4.1.3 Protocols for the counting measures

Weighted kappa statistic. The weighted kappa statistic can be written in terms of inner products too. Let us define the vectors $\mathbf{r}_{xv}$, with elements

$$(\mathbf{r}_{xv})_i = \begin{cases} 1 & \text{if } v_{xi} = v, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

We can write an element of the matrix p_{ax} in vector notation as

$$p_{ax}(v,w) = \frac{|\{i \in I_a \cap I_x | v_{ai} = v, v_{xi} = w\}|}{|I_a \cap I_x|} = \frac{\mathbf{r}_{av}' \mathbf{r}_{xw}}{\mathbf{r}_a' \mathbf{r}_x}.$$

An element of the matrix q_{ax} is written in vector notation as

$$q_{ax}(v,w) = \sum_{w'} p_{ax}(v,w') \sum_{v'} p_{ax}(v',w) = \frac{\mathbf{r}_{av}' \mathbf{r}_x \mathbf{r}_{xw}' \mathbf{r}_a}{(\mathbf{r}_a' \mathbf{r}_x)^2}.$$

Now construct a vector $\mathbf{p}_{ax}$ of the matrices by putting the columns behind each other. Then the observed value $O_{ax} = \mathbf{p}'_{ax}\omega$ and the expected value $E_{ax} = \mathbf{q}'_{ax}\omega$ where ω is a vector of the weights matrix. If we have the observed value and the expected value, then the kappa statistic can be calculated in the normal way.

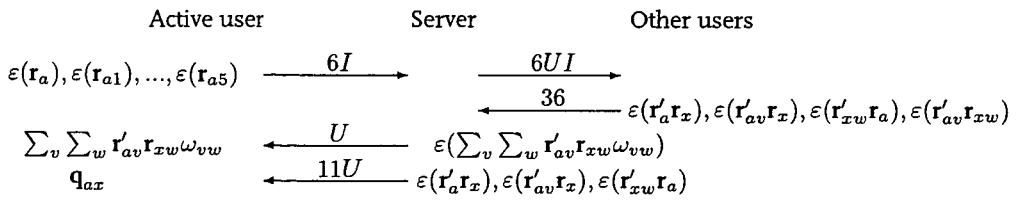

Figure 4.3: Protocol for the kappa statistic.

The protocol (Figure 4.3) makes use of the Paillier system as described in Section 3.2.2. The active user calculates the vectors $\mathbf{r}_a$ and $\mathbf{r}_{av}$ for $v = 1, \ldots, 5$. These vectors are encrypted and send to the other users. The other users apply an inner product protocol to the vectors. The encrypted inner products are sent to the server, the server calculates the encrypted sum $\varepsilon(\sum_v \sum_w \mathbf{r}'_{av}\mathbf{r}_{xw}\omega_{vw})$ via an inner product protocol and sends this sum and the remaining inner products $\varepsilon(\mathbf{r}'_a\mathbf{r}_x), \varepsilon(\mathbf{r}'_{av}\mathbf{r}_x)$ and $\varepsilon(\mathbf{r}'_{xw}\mathbf{r}_a)$ to the active user. The active user can calculate the inner product $\mathbf{p}'\omega$ and the vector $\mathbf{q}$. As we assume that the weight matrix ω is public, the user can calculate the inner product $\mathbf{q}'\omega$ and hence obtain the kappa statistic.

Majority voting. Recall that the majority voting similarity is

$$s(a, x) = (2 - \gamma)^{c_{ax}} \gamma^{w_{ax}},$$

where $c_{ax} = |\{i \in I_a \cap I_x | v_{xi} \in C(v_{ai})\}|$ and $w_{ax} = |\{i \in I_a \cap I_x | v_{xi} \notin C(v_{ai})\}|$. Define the vector $\mathbf{r}_{xv}$ as in (4.3), then we can rewrite c_{ax} as

$$c_{ax} = \sum_v \sum_{w \in C(v)} \mathbf{r}'_{av}\mathbf{r}_{xw},$$

and w_{ax} as

$$w_{ax} = \mathbf{r}'_a\mathbf{r}_x - c_{ax}.$$

The active user calculates his vectors $\mathbf{r}_a$ and $\mathbf{r}_{av}$ for $v = 1, \ldots, 5$. These vectors are encrypted and send to the other users, as shown in Figure 4.4. The other users apply an inner product protocol to the vectors. The encrypted inner products are sent to the server, the server calculates the encrypted sum $\varepsilon(\sum_v \sum_{w \in C(v)} \mathbf{r}'_{av}\mathbf{r}_{xw})$ via an SUM protocol and sends this sum and the remaining inner product $\varepsilon(\mathbf{r}'_a\mathbf{r}_x)$ to the active user. The active user can calculate c_{ax} and w_{ax}, and hence the majority voting similarity.

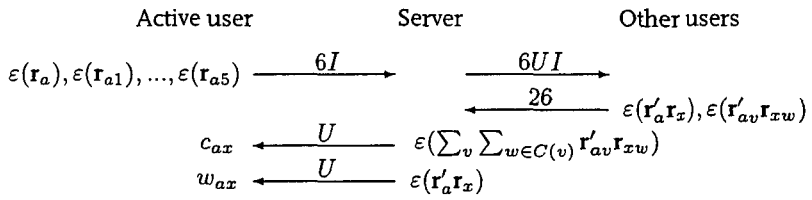

Figure 4.4: Protocol for the majority-voting similarity.

4.2 Protocols for the predictors

Standard predictor. Recall that the standard predictor is given by

$$p_{ai} = \bar{v}_a + \frac{\sum_{y\in U_i} s(a,y)(v_{yi} - \bar{v}_y)}{\sum_{y\in U_i} |s(a,y)|},$$

where the set U_i could be restricted to the users with a similarity above a certain threshold. Such a restricted set is called U'_i. Define the vector $\mathbf{s}_a$ as

$$s_{ax} = \begin{cases} s(a,x) & \text{if user } x \in U'_i, \\ 0 & \text{otherwise,} \end{cases} \tag{4.4}$$

Then the standard predictor is rewritten as

$$p_{ai} = \bar{v}_a + \frac{\sum_{y=1}^{U} s_{ay} w_{yi}}{\sum_{y=1}^{U} |s_{ay}| r_{yi}},$$

where w_{yi} is defined as in (4.1). As the user knows the similarities, he can construct a vector $\mathbf{s}_a$ and a vector $|\mathbf{s}_a|$ with elements $|s_{ax}|$. The active user encrypts this vectors and sends them to the server (Figure 4.5). The server sends the elements of the vectors to the appropriate users. The users calculate the encrypted multiplications $\varepsilon(s_{ay}w_{yi})$ and $\varepsilon(|s_{ay}|r_{yi})$. They can add a random term via (3.8) to protect their data even better. Subsequently, their messages are sent to the server. The server calculates the encrypted sums and sends them to the user. The user can decrypt the messages and obtain the prediction.

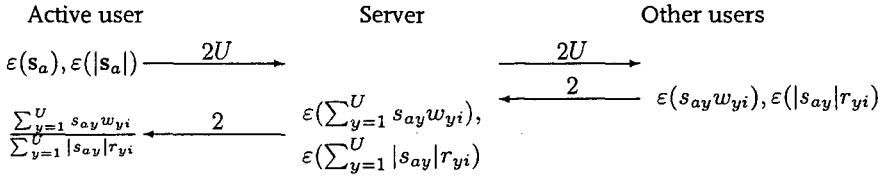

Figure 4.5: Protocol for the standard predictor.

Simple predictor. The simple predictor can be secured in an identical way as the standard predictor.

Majority voting predictor. The active user encrypts the elements $s(a,x)$ of the vector $\mathbf{s}_a$ and sends them to the server, as shown in Figure 4.6. The server sends the elements of the vector to the corresponding users. The other users calculate $\varepsilon(s(a,x)(\mathbf{r}_{xv})_i)$, for $v = 1,\ldots,5$, where $\mathbf{r}_{xv}$ is defined as in (4.3) and send it back to the server. The server collects the information and calculates $\varepsilon(\sum_x s(a,x)(\mathbf{r}_{xv})_i)$, for $v = 1,\ldots,5$. The user can decrypt these sums and calculates the prediction.

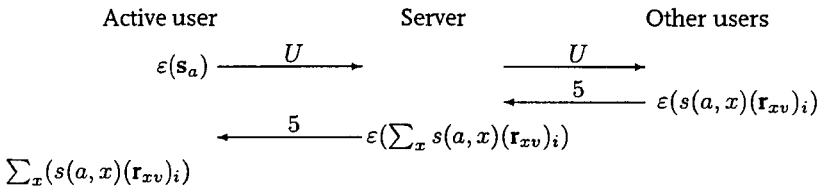

Figure 4.6: Protocol for the majority voting predictor.

Chapter 5

Protocols for the Item-Based Algorithm

In this chapter we shall derive a protocol for the item-based algorithm. We shall use the adjusted cosine (2.18) as similarity measure and the standard item-based predictor (2.19) as predictor. Other similarities, as described in Chapter 4 can be obtained in a similar way.

5.1 Protocol for the adjusted cosine measure

The adjusted cosine similarity is given by

$$s(i,j) = \frac{\sum_{x \in U_i \cap U_j} (v_{xi} - \overline{v}_x)(v_{xj} - \overline{v}_x)}{\sqrt{\sum_{x \in U_i \cap U_j} (v_{xi} - \overline{v}_x)^2} \sqrt{\sum_{x \in U_i \cap U_j} (v_{xj} - \overline{v}_x)^2}} = \frac{\sum_{x=1}^{U} w_{xi} w_{xj}}{\sum_{x=1}^{U} w_{xi}^2 \sum_{x=1}^{U} w_{xj}^2}, \tag{5.1}$$

where w is defined as in (4.1). Every user can calculate his own mean. The adjusted cosine consists of three sums over the users, in which each user can calculate his own part. If a user did not rate both items he can send a zero back. We can calculate the sums in two ways, the first of which is to use the threshold cryptosystem like in Section 3.3. The users calculate their part of the sums and encrypt them with the public key of the key-sharing scheme (Figure 5.1). The server collects the parts and computes the encrypted sums by multiplying the different parts. Then he sends the encrypted sums back to the users. The users can apply their secret share and send the result back to the server. If the server has enough decrypted shares, he can decrypt the three sums.

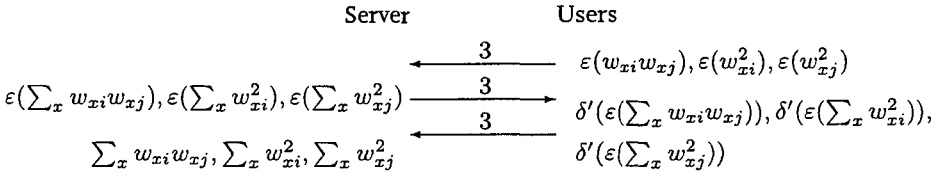

Figure 5.1: Protocol for the adjusted cosine using key sharing.

Another possibility is to use two servers instead of one, as shown in Figure 5.2. In that case, the parts of the users are encrypted with the public key of the decryption server. The encrypted parts are sent to the recommender server, who calculates the sums and sends them to the decryption server. The decryption server can decrypt the sums, and calculate the similarity between the items. This similarity is sent back to the recommender server. In order to work well, the two servers should be independent, as otherwise the messages of the users can be decrypted. The decryption server should be sure that the values given to it are encrypted sums, and not the encrypted messages of the users.

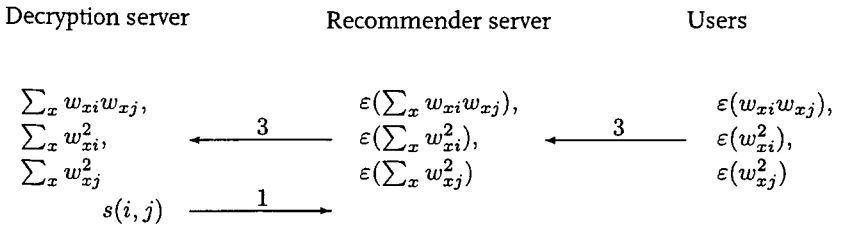

Figure 5.2: Protocol for the adjusted cosine with two servers.

5.2 Protocol for the predictor

Recall that the item-based predictor is given by

$$p_{aj} = \bar{v}_j + \frac{\sum_{i \in I_a} s(j,i)(v_{ai} - \bar{v}_i)}{\sum_{i \in I_a} |s(j,i)|}. \tag{5.2}$$

The item mean is a sum over the votes from all users divided by the number of users who gave their opinion on the item. The similarities between the items are stored at the server, just as the item means. We start with deriving a protocol for the calculation of the item means.

5.2.1 Protocol for the item mean

The item mean is a sum over the users, so for the protocol we can either use a threshold cryptosystem (Figure 5.3) or a two server system (Figure 5.4), just like in the protocol of the adjusted cosine. The users encrypt their vote v_{xi} and encrypt the indication they rated the item r_{xi}, with the public key of the key-sharing scheme. The server collects these values and calculates the encrypted sums. These sums are sent back to the users who can apply the secret share. The decrypted shares are sent back to the server. The server can decrypt the sum of the votes, and the number of users that rated the item. Hence, he has the mean of the item.

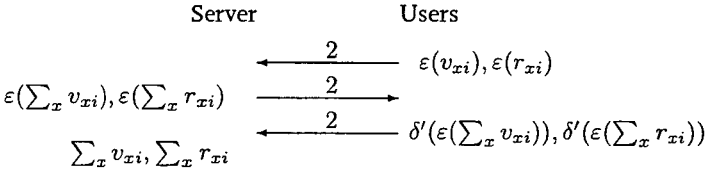

Figure 5.3: Protocol for the item mean using key sharing.

The other option is to use two servers. In this case the users encrypt with the public key of the decryption server, but send their encryptions to the recommender server. The recommender server calculates the encrypted sums and sends them to the decryption server. The decryption server decrypts the sums and calculates the item mean, which is sent to the recommender server.

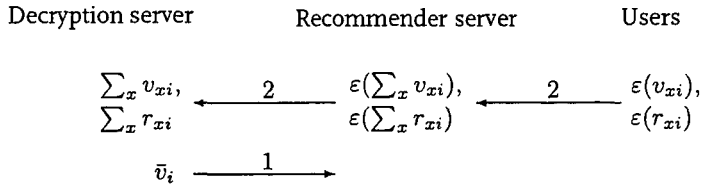

Figure 5.4: Protocol for the item mean with two servers.

5.2.2 Protocol for the standard item-based predictor

We can rewrite (5.2) into

$$p_{aj} = \frac{\bar{v}_j \sum_{i=1}^{I} r_{ai}|s(j,i)| - \sum_{i=1}^{I} r_{ai}s(j,i)\bar{v}_i + \sum_{i=1}^{I} s(j,i)v_{ai}}{\sum_{i=1}^{I} r_{ai}|s(j,i)|}.$$

The server has the knowledge of the similarities between the items, so it can calculate the encrypted normalization constant $\varepsilon(k) = \varepsilon(\sum_{i=1}^{I} r_{ai}|s(j,i)|)$, and send it to the user, see Figure 5.5. The server also calculates another encrypted constant $\varepsilon(M) = \varepsilon(\bar{v}_j k - \sum_{i=1}^{I} r_{ai}s(j,i)\bar{v}_i)$. The user encrypts his vector with votes and sends it to the server. The server calculates the inner product with the similarity vector $\mathbf{s}_j$. The constant M is added. The server sends the sum back to the user. The prediction is this sum divided by the normalization constant k.

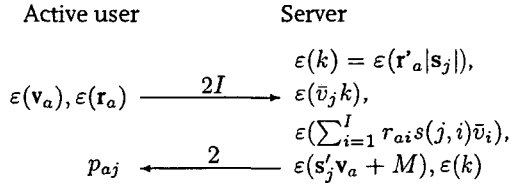

Figure 5.5: Protocol for the item-based prediction.

Chapter 6

Protocols for the Factor-Analysis Algorithm

A protocol for factor analysis must protect the user profiles and the model Λ, as it contains information about correlations between the items. As the user needs to use the matrix in an un-encrypted way, we decided to use a so-called personal server.

6.1 The personal server

The personal server is a piece of hardware and/or software installed in the user's device. It could for instance be installed in the internet radio player Streamium, which is developed at Philips Research. The personal server has the following properties.

- The personal server may know the information of its user.
- The personal server may not send information directly to the server, but only through the user.
- The personal server may know information about the central server.
- Only the central server decides which information on the personal server is given to the user.

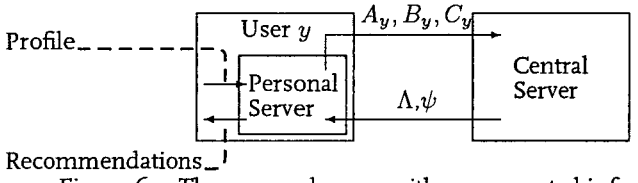

Figure 6.1: The personal server with un-encrypted information streams.

The factor-analysis algorithm has information streams as depicted in Figure 6.1. The server sends the model Λ, ψ to the personal server, and the user sends his profile to the personal server. With this information the personal server can calculate new predictions for the user and update information for the server. The update information A_y, B_y and C_y, as given in (2.22), is send via the user to the server. The server can compute $\sum_{y=1}^{U} A_y, \sum_{y=1}^{U} B_y$ and $\sum_{y=1}^{U} C_y$, which are used for the update of the model Λ, ψ. The central server decides how many recommendations a user receives. The security protocol for factor analysis can be split into two parts. The first part is the protocol for the model and the second part is the protocol for the user profile.

6.2 Protocol for the model

To avoid a lot of computations, we assume that the personal servers of all users have the same public key and private key. Therefore, model Λ and variance ψ are encrypted only once per iteration with the public key of the personal server (Figure 6.2). The personal servers are the only ones who can decrypt the model Λ and variance ψ. The user of the personal server does not own the private key, so he can not decrypt model Λ and variance ψ.

6.3 Protocol for the profiles

The protocol for the profiles makes use of the threshold Paillier system described in Section 3.3. The users send the encrypted update information A_y, B_y and C_y to the server, as shown in Figure 6.2. The server can calculate the encrypted sums $\varepsilon(\sum_y A_y), \varepsilon(\sum_y B_y)$ and $\varepsilon(\sum_y C_y)$ by multiplying the incoming information. These encrypted sums are sent back to the users. The users calculate the decrypted shares $\delta'(\varepsilon(\sum_y A_y)), \delta'(\varepsilon(\sum_y B_y))$ and $\delta'(\varepsilon(\sum_y C_y))$. The decrypted shares are sent back to the server. If the server has enough decrypted shares available, he can decrypt the messages and obtain $\sum_y A_y, \sum_y B_y$ and $\sum_y C_y$, which he uses next to update the model.

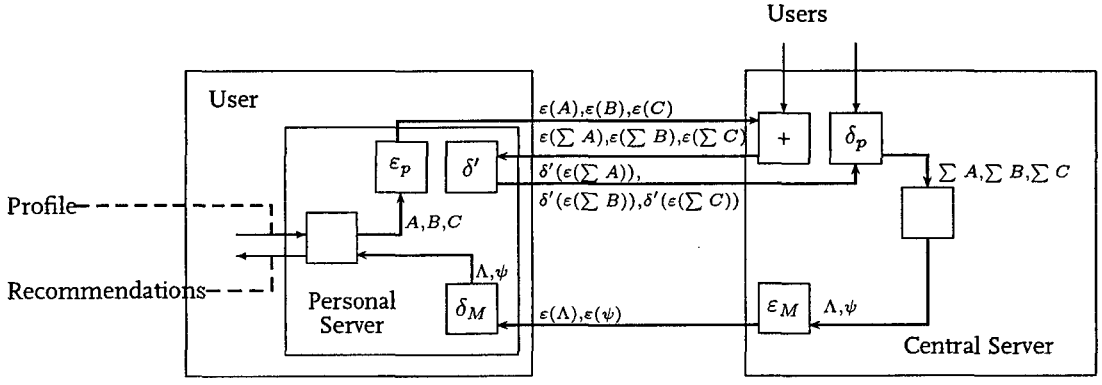

Figure 6.2: Protocol for factor analysis.

In a key-sharing scheme, the running time is increasing when the number of users increases (Chapter 8). Therefore we split big groups of users randomly into smaller groups, thus reducing the running time of the algorithm. The server can calculate with the protocol mentioned above the sums per group. The total of these sums is exactly the information the server needs. Instead of performing a key sharing protocol, we could also use a two server protocol, as shown in Figure 6.3. Then we encrypt the update information of the users with the public key of the decryption server, and send it to the recommender server. The recommender server can multiply the incoming information and sends the resulting encrypted sum to the decryption server. The decryption server has the secret key, so it can decrypt the result, which is sent back to the recommender server.

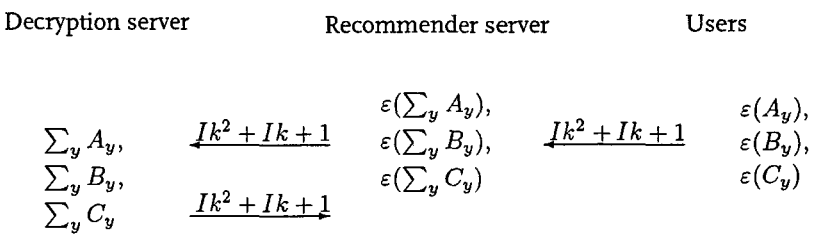

Figure 6.3: Protocol factor analysis with two servers.

Chapter 7

Factor-Analysis Recommendation Results

In this chapter we present the results of the factor-analysis algorithm run with the EasyAccess database on a Pentium II PC. In this Chapter, we will focus on quality of prediction and the running time of the algorithm without encryption. Chapter 8 deals with the running time of the algorithm with encryption.

7.1 Database

The EasyAccess database is the result of a user test with the Jukebox recommender system. It contains the user profiles of 508 users. The total number of songs is 5255. The database provides 97765 ratings (3.7 % of the possible ratings). So, the users voted on average 192 songs. In real life there will be more users. For instance the EachMovie database consists of 72916 users and 1628 movies. That database has 3% of the possible ratings.

7.2 Test setup

First the users who voted once were omitted from the database. These were 18 users. Then we chose at random 20% of the votes, which we kept apart as a test set. The remaining votes were used to build a factor-analysis model. The model was used to make predictions for the votes in the test set. As measurements for the error in the prediction we use the Mean Square Error (MSE), the Mean Absolute Error (MAE) and the Quality of prediction (Q), which are defined as

$$\begin{aligned} \text{MSE} &= \frac{1}{U}\sum_{y=1}^{U}\frac{\sum_{i\in S_y}(v_{yi}-p_{yi})^2}{|S_y|}, \\ \text{MAE} &= \frac{1}{U}\sum_{y=1}^{U}\frac{\sum_{i\in S_y}|v_{yi}-p_{yi}|}{|S_y|}, \\ \text{Q} &= \frac{4-\text{MAE}}{4}, \end{aligned}$$

where S_y denotes a set with votes of user y. We write MSE_{test}, MAE_{test} and Q_{test} if the test set from user y is used as S_y, and $\text{MSE}_{\text{model}}$, $\text{MAE}_{\text{model}}$ and Q_{model} if we use the model set as S_y. With the last measures, we can show to which extent the model fits the data. For comparison reason, we also calculated the errors of always predicting the middle score 3, which gives MAE = 1.29 and Q = 0.68, where $S_y = I_y$ for all users y. The MAE varies from 0 to 4, where a 0 indicates a perfect model. The MSE varies from 0 to 16, where 0 indicates a perfect model too. The quality of prediction Q varies from 0 to 1, where a 1 indicates a perfect model. In the subsequent section, the quality of prediction

Q_{test} of the test set will be our most important measurement for the effectiveness of the factor-analysis algorithm.

7.3 Quality of prediction of the factor-analysis algorithm

The test was repeated four times with different test sets. The mean values are given in Table 7.1. We calculated the error in the prediction for different classes k. This is equal to the number of columns (basis vectors) of the matrix Λ.

k	2	3	4	5	6	7
MSE_{model}	0.58	0.47	0.36	0.30	0.25	0.20
MAE_{model}	0.53	0.47	0.40	0.36	0.32	0.28
Q_{model}	0.87	0.88	0.90	0.91	0.92	0.93
MSE_{test}	0.93	0.93	0.94	1.06	1.04	1.09
MAE_{test}	0.67	0.66	0.65	1.00	0.68	0.68
Q_{test}	**0.833**	**0.835**	**0.837**	**0.833**	**0.831**	**0.828**
ψ	0.59	0.47	0.39	0.33	0.28	0.21
Time/it(sec)	4	6	9	15	20	26
Nr Iterations	13	17	18	19	20	19

Table 7.1: Prediction results of factor analysis for different number of classes k.

The optimal class for the EasyAccess database is $k = 4$, as the quality of prediction Q_{test} is big (Table 7.1). However the difference with the other numbers of classes is very small, as $k = 2$ and $k = 3$ are optimal for MSE_{test}. For clarity, we also show the quality of prediction QN against the number of classes k in Figure 7.1.

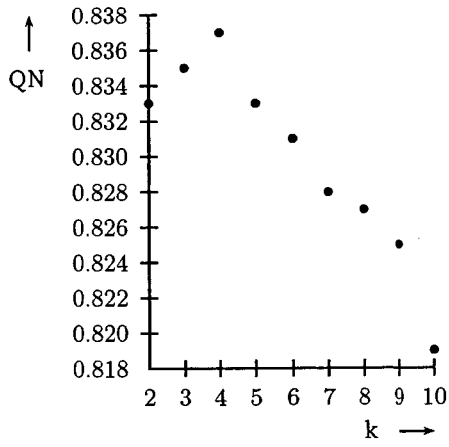

Figure 7.1: The quality of prediction against the number of classes.

The Pearson correlation, constrained Pearson correlation, kappa variance (a variant of the kappa statistic) and the weighted kappa, were tested with the EasyAccess database [15] too. Similarities between users were only calculated when they had at least 5 or 10 items in common, so users with fewer than 5 or 10 items could not receive recommendations. Three different similarity thresholds 0.3, 0.4 and 0.5 were used. This resulted in six tests for each similarity metric. We simply took the average Q_{test} and number of predictions over these six tests. We took the average Q_{test} for the factor-analysis algorithm over the number of classes $k = 2, \ldots, 7$. This average prediction quality Q_{test} is plotted

against the number of predictions in Figure 7.2. The factor-analysis algorithm can make predictions for more users than the algorithms based on the Pearson correlation or the kappa statistic, while the quality of prediction is as good as the quality of prediction of the algorithms based on the kappa statistic.

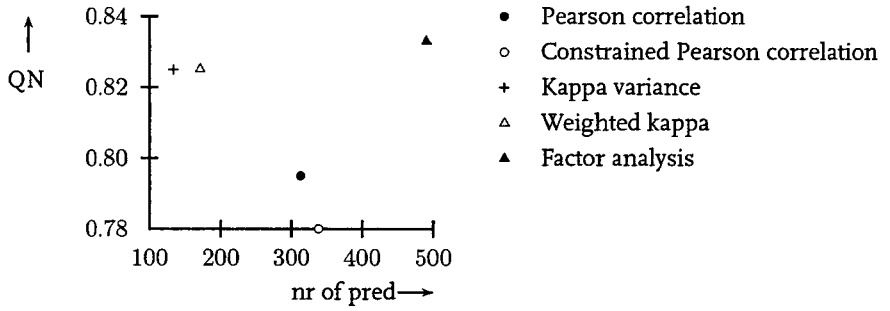

Figure 7.2: The quality of prediction against the number of predictions.

7.4 Running time of the factor-analysis algorithm

As we mentioned earlier, the time complexity of the factor-analysis algorithm is $O(UIk^2)$. In Figure 7.3 we see that the time per iteration of the algorithm rapidly increases when the number of classes is growing. The EasyAccess database is quite small with about 500 users and 5,000 items. If the database consists of 10,000 users, 10,000 items and we choose $k = 10$ then an iteration will approximately cost half an hour.

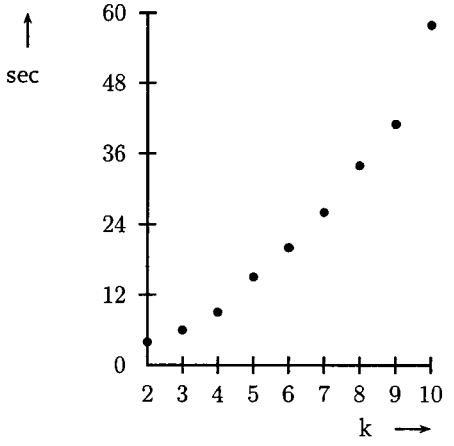

Figure 7.3: The time per iteration (sec) against the number of classes.

7.5 Varying the minimum number of votes required to obtain predictions

In the first test we only looked at the users with two or more votes, as we deleted the users with only one vote. Next, we delete users with fewer than 5 votes, 10 votes, 15 votes and 20 votes. The optimal number of classes is labeled k-Opt. Unlike what we expected, when we delete users with a small number of votes the prediction quality does not really increase, as shown in Table 7.2.

MinVotes	k-Opt	MSE_{model}	MAE_{model}	Q_{model}	MSE_{test}	MAE_{test}	Q_{test}	ψ	U	I
2	4	0.36	0.40	0.90	0.94	0.65	0.837	0.39	489	5249
5	3	0.48	0.48	0.88	0.91	0.65	0.837	0.49	465	5251
10	3	0.49	0.48	0.88	0.92	0.65	0.838	0.51	441	5246
15	3	0.52	0.51	0.87	0.88	0.66	0.836	0.56	404	5252
20	4	0.42	0.45	0.89	0.99	0.69	0.827	0.48	385	5251

Table 7.2: Varying the minimum number of votes.

7.6 Update the model

Next, we investigate the effect of incrementally calculating the model, where we choose $k = 4$. We build a model with 50% of the data. Then we add 5% of the data and use the model of 50% as initialization. We can compare the resulting model with a model build with 55% of the data and a random initialization. If we look at the quality of prediction in Table 7.3 and 7.5 we see that the update method works just as good as the model with random initialization. We also did the update with the minimum number of two iterations (Table 7.4). The predictions with this method are as good as the model with random initialization too. In Figure 7.6 the quality of prediction for the update methods and the model built from random initialization is plotted against the percentage of the data set we used. As we already saw in the tables, the update models are as good as the model built with a random initialization.

Perc. of the dataset	50	55	60	65	70	75	80
MSE_{test}	1.18	1.12	1.08	1.0	0.95	0.92	0.88
MAE_{test}	0.73	0.72	0.70	0.67	0.66	0.64	0.63
Q_{test}	0.817	0.821	0.826	0.832	0.836	0.839	0.841
Nr iterations	21	5	5	4	4	4	3

Table 7.3: Update the model until convergence.

Perc. of the dataset	50	55	60	65	70	75	80
MSE_{test}	1.18	1.13	1.09	1.01	0.96	0.91	0.88
MAE_{test}	0.73	0.72	0.70	0.68	0.66	0.64	0.63
Q_{test}	0.817	0.821	0.825	0.831	0.835	0.839	0.842
Nr iterations	21	2	2	2	2	2	2

Table 7.4: Update the model with two iterations.

Perc. of the dataset	50	55	60	65	70	75	80
MSE_{test}	1.18	1.08	1.08	1.04	0.97	0.97	0.93
MAE_{test}	0.73	0.7	0.7	0.68	0.66	0.66	0.65
Q_{test}	0.817	0.825	0.825	0.829	0.835	0.835	0.838
Nr iterations	21	20	21	18	19	17	16

Table 7.5: Model built with random initialization.

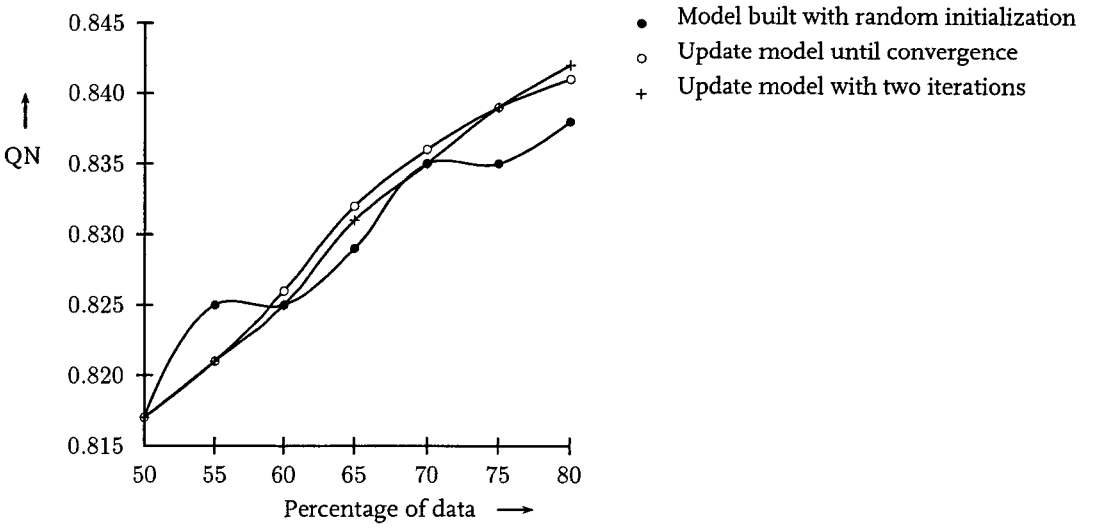

Figure 7.4: Quality of prediction for different number of iterations.

Chapter 8

Encryption Time Results

The security protocol for factor analysis was implemented in Java and tested on a Pentium II system too. The model Λ, ψ is encrypted by the server and sent to the user. The user calculates the matrices A, B and the number C. The number of messages to encrypt or decrypt for the server and one user are shown in Table 8.1.

Phase	Server	User
Encryption model	$Ik+1$	
Decryption model		$Ik+1$
Encryption A, B, C		Ik^2+Ik+1
Share decryption sums		Ik^2+Ik+1
End decryption sums	Ik^2+Ik+1	

Table 8.1: Number of elements per calculation phase.

8.1 Results for the Paillier system

We implemented the Paillier system of Section 3.2.2 in Java, and tested it on a Pentium II computer. The Paillier system only depends on the number of messages that are encrypted or decrypted, so in our case it depends on I and k. We measured the running time of the algorithm when we encrypted an $I \times k$ matrix, and when we decrypted an $I \times k$ matrix. The results are shown in Table 8.2, where we see that the algorithm is linear in both I and k.

I	100		200		5255	
k	2	10	2	10	2	10
Encryption	2	7	3	14	67	333
Decryption	1	6	3	12	63	314

Table 8.2: Encryption and decryption times in seconds for the Paillier system of Section 3.2.2.

8.2 Results for the threshold Paillier system

The threshold Paillier system, as described in Section 3.3 was implemented in Java too, and tested on a Pentium II computer. This system also depends on the number of messages that are encrypted and decrypted, but it depends on the number of users per group U_g and the threshold t too. In Tables 8.3

and 8.4 we encrypted and decrypted an $I \times k$ matrix, where $I = 100$ and $k = 2$. First, the threshold was chosen $t = 5$, while we investigated the running time in seconds with different number of users per group U_g. Next, the number of users per group was chosen $U_g = 50$, while we investigated the running time in seconds with different values of the threshold t.

U_g	25	50	75	100	125	500
Encryption	2	2	2	2	2	2
Share decryption	3	4	5	6	7	28
End decryption	4	8	14	21	25	148

Table 8.3: Encryption and decryption times in seconds for the threshold Paillier system of Section 3.3 and different values of U_g.

The encryption in the threshold system is the same as the encryption in the preceding section, so it does not depend on U_g or t, as shown in Tables 8.3 and 8.4. The share decryption is the running time of the share decryption algorithm for one user (one $I \times k$ matrix). In Table 8.3 is shown that the share decryption is linear in the number of users per group U_g. The share decryption does not depend on the threshold t, as shown in Table 8.4. The end decryption is the running time of the end decryption algorithm for one $I \times k$ matrix. The end decryption is both linear in t and U_g as shown in Table 8.4 and 8.3.

t	2	5	10	15	20	50
Encryption	2	2	2	2	2	2
Share decryption	4	4	4	4	4	4
End decryption	3	9	17	27	36	103

Table 8.4: Encryption and decryption times in seconds for the threshold Paillier system of Section 3.3 and different values of t.

8.3 Estimated running time of the factor-analysis algorithm with encryption

In order to get new recommendations once a week, the encryption algorithm should not cost too much time. For the EasyAccess database it holds that $I = 5255$, $k = 4$ and $U = 500$. For the security of the user profiles we make the choice that 10% of the users are needed to decrypt a message, as not all the users are always online. The threshold Paillier system depends on the threshold t and the number of users per group U_g. Therefore it is better to not simply use $U_g = 500$ and $t = 50$, but to divide the users into smaller groups and first calculate sums over the groups. We can make 5 groups of 100 users each, where the sum of each group can be decrypted if 10 users are available. Another option is to use 10 groups with 50 users each and a threshold of 5 users.

$I = 100, k = 2$	$U_g = 500, t = 50$	$U_g = 100, t = 10$	$U_g = 50, t = 5$
Encryption	2	2	2
Share decryption	28	6	3
End decryption	1376	39	8

Table 8.5: Encryption and decryption times in seconds for the threshold Paillier system of Section 3.3 with different numbers of users per group U_g and different values of the threshold t.

In Table 8.5 we encrypted and decrypted an 100×2 matrix for different values of the threshold t and different values for the number of users per group U_g. The encryption, share decryption and end

decryption are only once applied on this matrix. The table shows that it is indeed faster to divide the users in groups.

Phase	$U_g = 500, t = 50$	$5 * (U_g = 100, t = 10)$	$10 * (U_g = 50, t = 5)$
Encryption model (Server)	134	134	134
Decryption model (One user)	126	126	126
Encryption A, B, C (One user)	670	670	670
Decryption sums (Server)	630	630	630
Share decryption sums (One user)	14714	3153	1577
End decryption sums (Server)	723088	102473	42040
Iteration sequential two-server	4.5 days	4.5 days	4.5 days
Iteration parallel two-server	0.5 hour	0.5 hour	0.5 hour
Iteration sequential threshold	21.5 days	7.5 days	6 days
Iteration parallel threshold	8.5 days	30 hours	12 hours

Table 8.6: Estimated running time of an iteration factor analysis with encryption, where the running time is in seconds, unless stated otherwise.

In Chapter 7 we showed that it is possible to update the model with only 2 iterations. In Table 8.6, the running time of one iteration of factor analysis is estimated. We estimated the running time for a sequential and parallel factor-analysis algorithm with two servers and for a sequential and parallel factor-analysis algorithm with key sharing. With a sequential algorithm, we mean that a user can only start his calculations if the preceding user is finished. In a parallel algorithm the users can do their calculations simultaneously. In real life, the running time will be comparable (but a bit longer) than a parallel algorithm. A two-server model is the fastest and has a high probability of updating the model within a week. With the threshold decryption and the users divided into small groups it is also possible to update the model within a week. For a database with 10,000 users, 10,000 items and 10 classes, the two-server model with one iteration (parallel) will cost 5 hours, (the sequential 2.5 years). The threshold system costs 4 months for one iteration parallel, and 3.5 years sequential. A security system where one server collects the information and the other server decrypts the sums seems therefore the best solution. The threshold system can be applied to a large database if the calculations for the share decryption are executed in parallel. In small databases the threshold system works well. However, the running time of the algorithms could be reduced by using another implementation. It is also possible to improve the running time by doing calculations of the server in parallel.

Chapter 9

Conclusion

Collaborative filtering systems are developed in order to relieve the problem of searching through all the information for interesting items. The standard collaborative filtering systems still have some flaws.

- The server stores the ratings of the users in a large database. This is very personal information that should be protected against unwanted use.
- The standard user-based algorithm calculates similarities based on items the users both rated. As users rate only a small part of the available items, it is difficult to find pairs of users with enough rated items in common. A lot of people are therefore not able to receive recommendations.
- The standard user-based algorithm requires a computation that grows quadratic in the number of users.

In Chapter 2 we described the user-based algorithm and its variants, the item-based algorithm and the factor-analysis algorithm. User-based algorithms use a database with votes to calculate similarities between users. New predictions are obtained by interpolation in which the similarities act as weights. Item-based algorithms work in a similar way, but now the algorithm calculates similarities between items. The factor-analysis algorithm first builds a model from the database with votes, which is then used for calculating predictions. Chapter 3 introduces a cryptosystem, with which we develop security protocols for the user-based algorithms (Chapter 4), item-based algorithms (Chapter 5) and factor-analysis algorithm (Chapter 6).

As the factor-analysis algorithm has the best time complexity, it was decided to test it with the EasyAccess database. In Chapter 7 it was shown that factor analysis can calculate recommendations for more users than algorithms based on the Pearson correlation or kappa statistic, without having a loss in the quality of prediction. Small numbers of classes give already good predictions. Usually, 20 iterations were necessary to build a new model. However, if an old model was used as initialization, the model could be updated with only a few iterations. Furthermore, factor analysis is able to calculate recommendations for a lot of users in a reasonable time.

The factor-analysis algorithm was tested with encryption too. In Chapter 8 it was shown that encryption and decryption has a great impact on the running time of the algorithm. For the EasyAccess database it is still possible to update the model once a week with both the threshold system and the two server system; for larger databases only the two-server system is an option, since the threshold system is too slow. However, the running times for both systems can be improved by using another implementation or doing more calculations in parallel. A collaborative filtering system with encryption should take care that the number of elements, that need encryption is not too large, as it takes longer to make recommendations.

Bibliography

[1] Charu Aggarwal, Joel Wolf, Kun-Lung Wu, and Philip Yu. Horting Hatches an Egg: A New Graph-Theoretic Approach to Collaborative Filtering. *Proceedings of the ACM KDD'99 Conference*, pp. 201–212, 1999.

[2] D. Billsus and M. Pazzani. Learning Collaborative Information Filters. *Proceedings of Recommender Systems Workshop*, pp. 46–54, 1998.

[3] John Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 43–52, July 1998.

[4] J. Canny. Collaborative Filtering with Privacy via Factor Analysis. *Proceedings of ACM SIGIR'02*, pp. 238–245, 2002.

[5] J. Canny. Collaborative Filtering with Privacy. *Proceedings of the IEEE Security and Privacy Conference*, pp. 45–57, 2002.

[6] R. Cramer, R. Gennaro, and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. *European Transactions on Telecommunications*, Vol. 8(5): pp. 481–490, 1997.

[7] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, Vol. 41(6): pp. 391–407, 1990.

[8] A. Dempster, N. Laird, and D. Rubin. Maximum Likelihood from Incomplete Data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39: pp. 1–38, 1977.

[9] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing Decryption in the Context of Voting or Lotteries. *Proceedings of the 4th International Conference on Financial Cryptography, Lecture Notes in Computer Science*, Vol. 1962, pp. 90, 2000.

[10] T. El Gamal. A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, Vol. 31(4) : pp. 469–472, 1985.

[11] D. Greening. Building Consumer Trust with Accurate Product Recommendations. *Likeminds White Paper LMWSWP-210-6966*, 1997.

[12] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. *Proceedings of ACM SIGIR'99*, pp. 230–237, 1999.

[13] George Karypis. Evaluation of Item-Based Top-N Recommendation Algorithms. *Proceedings of the 10th Conference on Information and Knowledge Management*, pp. 247–254, 2001.

[14] Natasha Kravtsova. A Recommender System for CE based on Collaborative Filtering. Technical report TN 2001/258, Philips Research, 2001.

[15] Natasha Kravtsova. Improvements in the Collaborative Filtering Algorithms for a Recommender System. Technical report TN 2001/542, Philips Research, 2002.

[16] A. Nakamura and N. Abe. Collaborative Filtering using Weighted Majority Prediction Algorithms. *Proceedings of the 15th international conference on machine learning*, pp. 395–403, July 1998.

[17] Pascal Paillier. Public-key Cryptosystems based on Composite Degree Residuosity Classes. *Advances in Cryptology- EUROCRYPT'99, Lecture Notes in Computer Science*, Vol. 1592:pp. 223–238, 1999.

[18] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An Open Architecture for Collaborative Filtering of Netnews. *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, pp. 175–186, 1994.

[19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of Recommendation Algorithms for E-commerce. *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pp. 158–167, 2000.

[20] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of Dimensionality Reduction in Recommender System: A Case Study. *ACM WebKDD Web Mining for E-commerce Workshop*, 2000.

[21] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based Collaborative Filtering Recommendation Algorithms. *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pp. 158–167, 2000.

[22] L. Saul, and M. Rahim. Maximum Likelihood and Minimum Classification Error Factor-Analysis for Automatic Speech Recognition. *IEEE Transactions on Speech and Audio Processing*, Vol. 8(2): pp. 115–125, 2000.

[23] H.J.A. Schouten. Measuring Pairwise Interobserver Agreement When all Subjects are Judged by the Same Observers. *Statistica Neerlandica*, Vol. 36(2): pp. 46–61, 1982.

[24] A. Shamir. How to Share a Secret. *Communications of the ACM*, Vol.22(11): pp. 612–613, 1979.

[25] U. Shardanand and P. Maes. Social Information Filtering: Algorithms for Automating Word of Mouth. *Proceedings of CHI'95*, pp. 210–217, 1995.

[26] John Uebersax. Diversity of decision-making models and the measurement of interrater agreement. *Psychological bulletin* Vol. 101, pp. 140–146 http://ourworld.compuserve.com/homepages/jsuebersax/kappa.htm, July 2002.

Index