Outsourcing Financial Advice Keeping The Full Confidentiality of Client Data

André Bertolace[1] and Esther Hänggi[2]

[1]Ex Indiciis
[2]HSLU

October 9, 2020

Abstract

Privacy, confidentiality and data protection are and will remain core competencies of Swiss banking. In fact there exists a number of legal provisions that ensure the Bank-client confidentiality. The Federal Act on Data Protection (FADP), the Article 47 of the banking act and the EU's new General Data Protection Regulation (GDPR) are just a few of the many regulations that exist to ensure data protection.[1] Yet, embracing digital innovation and quickly reacting to new IT developments are critical to the survival of the banking sector in the years to come. This includes the use of cloud services, third party services providers and external data processors more often than before. The banks, as data curators, remain responsible for the privacy and the protection of the client's data. How to comply with the data protection requirements and still collaborate with external data processors? In this extended abstract we apply homomorphic encryption to solve this problem. We propose a concept for a recommender system for financial advise based on client's data while keeping this data fully confidential from the external data processor. In a proof of concept we show how to implement such a privacy-preserving recommender systems for financial applications and also show where there are the current technological limits.

1 Introduction

In e-commerce, the topic of recommender systems gained increasing importance in the late 90s. Digitally native companies recognized early on the unprecedented opportunities for personalization that came up with the Internet. No other channel ever before provided that easiness to collect data and a communication channel that could be employed to recommend products in such an unobtrusive way. Fast-forwarding to now-a-days, it is said that 35% of purchases on Amazon and 75% of content streamed on Netflix are results of recommendations based on algorithms.[2]

That said, these systems need to analyze large amount of personal information in order to give useful recommendations. This raises the question how to keep the data confidential while still using the advantages of data analysis to give the client individually tailored advice for investments.

[1]https://www.swissbanking.org/en/topics/information-for-private-clients/privacy-and-data-protection
[2]https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers

Table 1: How to collaborate in a secure manner? There are many alternatives, some more, other less secure

Method	Type	Comment	Verdict
NDA	Legal	No technical guarantee	
Anonymization/ Sanitization[a]	Technical	Data loss and vulnerable to external information attacks	
Encrypted	Technical	No data loss and secure[b]	

[a] k-anonymity, l-diversity, synthetic data, ...

[b] *assuming that the unencrypted data is kept in a protected environment*

One possibility is to rely on the legal framework to ensure confidentiality in a contractual form. This is not always sufficient since it does not technically prevent access to the data in question. Another alternative is to anonymize or sanitize the data. It is a technical solution but it introduces data loss and is vulnerable to external information attacks. Finally, one could use encryption methods to enforce data privacy.

A technical solution to this challenge is *homomorphic encryption* — a way to encrypt data that allows the data processing to be done *exclusively* on encrypted data, thereby permitting the secure outsourcing of data processing without giving the data processor access to the customer's data. The mathematical theory to do this and even some programming libraries already exist. This comes at a price of increased computational requirements. The concept is, therefore, not yet in wide use.

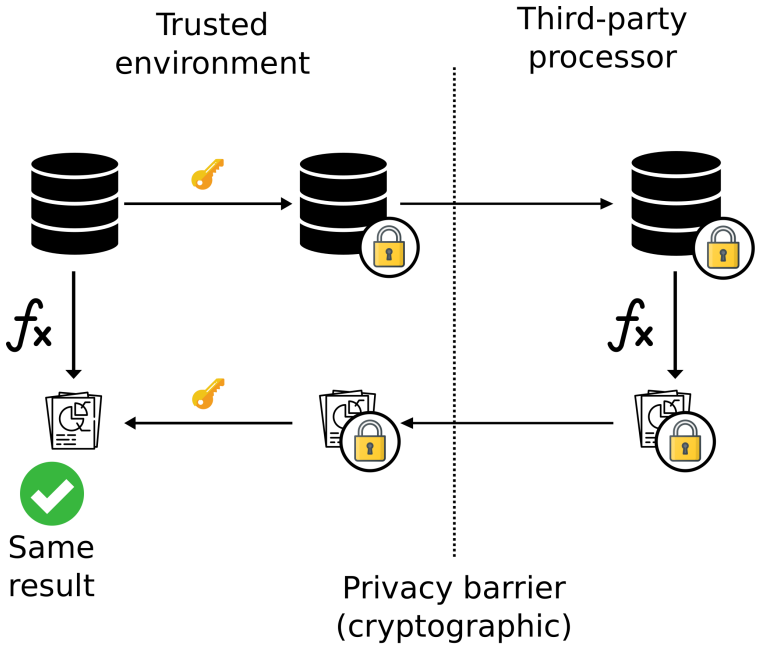

Figure 1: Homomorphic encryption: Analytics on encrypted data returns the same result as in unencrypted data

Since not all type of computation can be done on encrypted data, a recommender system

must be adapted for this technique to be applicable. We give such possible adaptations here and implement a proof-of-concept implementation of some textbook algorithms for recommender systems.

In Section 2 we describe which type of data we need to secure and the exact setup we consider, in Section 3 we introduce homomorphic encryption — the mathematical tool we will use to reach our goal. In Section 4 we give the details of our proposed solution using homomorphic encryption and in Section 5 we describe our experimental results of the proof-of-concept implementation. Finally, in Section 6 we draw the conclusions and describe our next steps.

2 Data Confidentiality in the Context of Recommender Systems

As usual in machine learning systems, recommender systems also have a training and a prediction steps. In our case, the training data could consist of all investments the clients have made up to a certain point in time. In the prediction step, the trained recommendation algorithm is used to calculate a specific recommendation for a given data set — e.g., the data of a user to which a financial product shall be recommended.

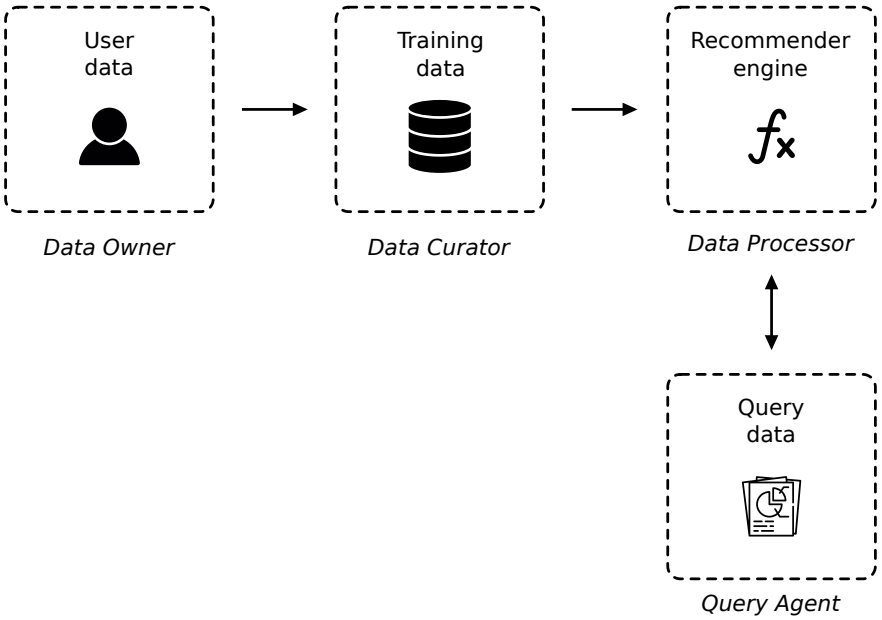

Figure 2: A schematic description of a recommender system.

Several entities are involved: the client, called *data owner*; the financial institution which provides financial advise and investment solutions called, for our purpose, as *data curator*; the external data-processor providing the recommender system is called the *data processor*; and the one who finally uses the trained mode, called here the *query agent*.

We conceive 3 different scenarios depending on the roles of the players:

1. The *data curator* develops a recommendation system in-house, using the data from the *data owners*, but would like to make it available *as a service* to external clients or if the developed

algorithm is not considered sensitive, since it only contains aggregated data. In this case *the query and the recommendation must be kept confidential*.

2. The recommendations shall be outsourced, but the algorithm is still considered to be sensitive. The actual development of the algorithm, i.e., the training, would then need to be performed by the *data curator*. In this case *the query, the recommendation and the algorithm shall be kept secret*.

3. The recommendation system is fully outsourced to a specialized firm — as it is the case in other industries. In this case *the query, the recommendation, the algorithm and the algorithm training are considered sensitive*

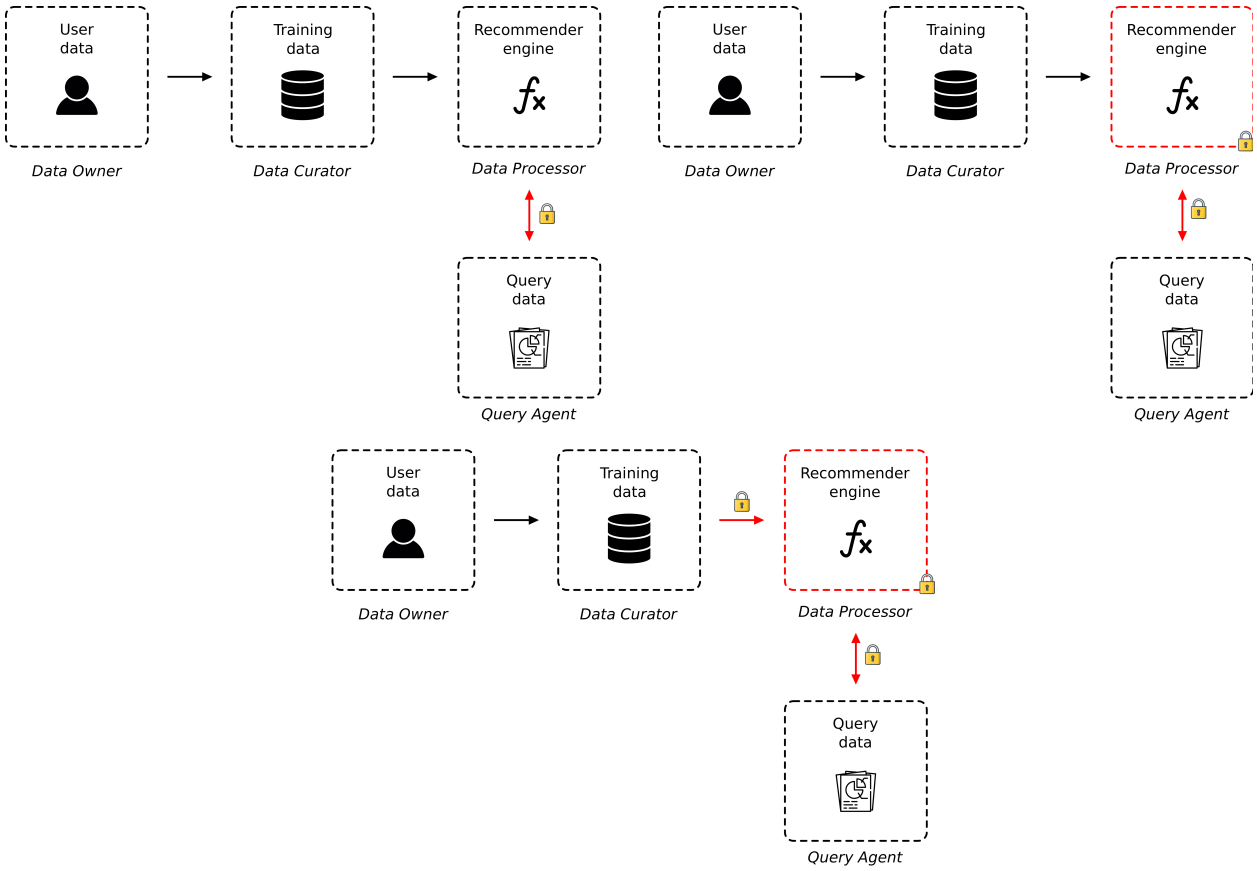

Figure 3: The three scenarios with the different privacy goals. Depicted in red is the data that needs to be kept private and, therefore, must only be available in encrypted form.

IBM and Banco Bradesco S.A. published a Proof-of-Concept [10] which falls on the first scenario: The bank wanted to implement a fraud detection mechanism based on data from both the retail department and the loan department of the bank. However, these two departments are legally not allowed to share data.

We will consider the third scenario, that is, where the query, the recommendation, the algorithm and the algorithm training are considered sensitive. The data processor shall provide all services

related to data analysis and the development of a recommender system, from the development of the algorithm, i.e., training the data, to the calculation of the actual recommendation.

3 Homomorphic Encryption

Homomorphic encryption is the affirmative answer to the question 'Is it possible to delegate processing of your data without giving away access to it?' [6]. An encryption scheme which is homomorphic, allows to encrypt data, then do operations on the encrypted data and obtain the encrypted result. This allows to securely outsource data processing, since the data processor can only be given encrypted data.

Many of the encryption schemes currently in use, such as RSA [11] are homomorphic for *some* operations, i.e. either addition or multiplication are possible to do on encrypted data, but not both. For a long time it was unclear whether there exist schemes which can do both addition and multiplication, an therefore implement any binary circuit, on encrypted data.

In 2009, Gentry [5] proposed the first fully homomorphic encryption scheme. The computational overhead of the scheme was gigantic, but it showed that such schemes are indeed possible. Since the first proposal by Gentry [5], these schemes have been further developed. The currently most widely used schemes are based on BFV [4, 1], CKKS [3], BGV [2] and GSW [7]. The computational overhead remains large, but has been brought down to a level where these schemes can be implemented and run.

Practical libraries now exist for these schemes. Most libraries are implemented in the programming language C++ for performance reasons, but others in Go or Haskell equally exist.

We will in the following use the BFV-scheme [4, 1] with the open source library SEAL — Microsoft SEAL [12] with our own Python wrapper (also available as an open source code).

While fully homomorphic encryption schemes can in principle implement any binary circuit, and therefore any computation as long as it is expressed in this way, they come with several limitations or restrictions depending on the scheme.

- Some schemes operate only on integers, e.g. the BFV-scheme [4, 1] other schemes also operate on floating point numbers, e.g., the CKKS-scheme [3]. This is not necessarily a big restriction since it is often possible to multiply everything with a large number and thereby obtaining expressions in integers. Nevertheless, this requires some analysis and makes some schemes more suitable for certain applications than others.

- The schemes can calculate addition and multiplication of encrypted numbers, or, more generally, calculate a polynomial of bounded degree on encrypted inputs. However, they cannot (directly) implement division, absolute values, taking the (square) root, exponentials or the calculation of trigonometric functions. The reason is that the algebraic structure they operate on is not closed under these operations (i.e., a scheme that operates on integers cannot implement division). This is a very serious restriction which seriously limits the calculations that can be done in a 'homomorphic way' in practice.

- Similarly, the schemes cannot implement discrete functions such as maximizations or implement computations with if-constructs. This would in fact violate the confidentiality of the input since observing which branch a program took would allow to gather information about the input. This is again a very serious restriction since most computations including in the

context of data analysis or recommender systems contain such constructions. One solution is to compute all possible branches, but this is again costly in terms of computing time.

- The schemes have several parameters which need to be chosen before the calculation, such as the modulo or noise properties. If the calculations include operations on large numbers, e.g. the modulus need to be chosen larger, thereby slowing down all calculations. Choosing these parameters wisely and optimizing them for the specific problem at hand can therefore speed up the calculations significantly.

- The schemes operate using a so-called *noise budget*. The different operations use up the noise budget and once it is used up a 'bootstrapping' procedure needs to be performed, which is time-costly. For most schemes, additions are 'cheap' in terms of noise budget, while multiplications are expensive. Again, optimizing the way calculations are done can, therefore, speed up the calculations significantly.

In brief, not all function can be implemented on encrypted data and the choice of scheme, the specific parameters and the calculation can be optimized to reduce the computational cost, which might otherwise be prohibitively high.

4 Reaching Confidentiality Against the Data Processor Using Homomorphic Encryption

We propose to include an encryption and decryption layer between the data curator and the data processor as depicted in Figure 4 to ensure that the data processor only receives encrypted data. More precisely, the data that the data processor receives to train the algorithm is encrypted using homorphic encryption. The data processor now designs the algorithm in encrypted form. When the record owner or data curator makes a query to receive a recommendation, the query is also encrypted and is evaluated by the encrypted algorithm. The data processor returns the result in encrypted form to the data curator who decrypts it.

As discussed in Section 3, functions such as division or taking the maximum over several entries are not possible to implement in homomorphically encrypted form. Since most recommender systems make heavy use of these functions, they must be adapted before they can be used in this setup. In [8], Graepel, Lauter and Naehrig therefore propose to make several modifications to a recommender system so it can be expressed as a polynomial function of degree at most D:

- Divide the recommender system into two parts (functions) f and g such that the recommender system is $f \circ g$ and f can be calculated homomorphically, while g is a discretization function which must be applied locally (by the client). For example, the calculation could be such that f outputs a real value or integer, which is then discretized by the client by only taking the sign of the result. Many functions can be divided in this way and we will use this system in our example.

- Use the Taylor series up to order D to approximate a function. This normally works well to approximate exponentials or trigonometric functions.

In addition to this, we also need to avoid functions, such as division, which we can do by avoiding specific renormalizations.

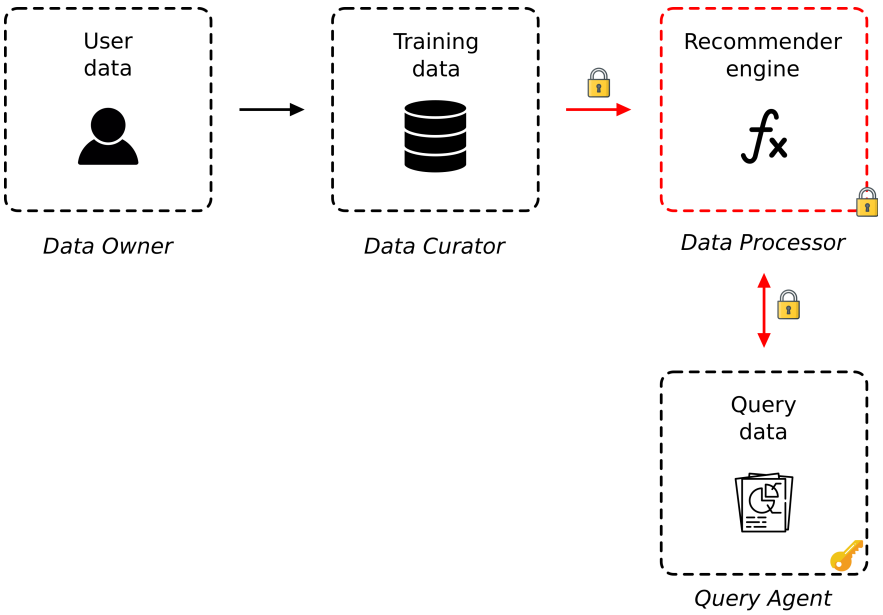

Figure 4: The training data and query are encrypted before the data analysis. The data processor therefore only receives encrypted data. The answer to the query is returned in encrypted form and needs to be decrypted to obtain the recommendation.

Below, we apply these modifications to recommender systems based on 'neighbourhood-based collaborative filtering' in order to implement them in a homomorphic encryption scheme. We assume that we start from a database in matrix form which contains a rating per user and item. Recommender systems based on 'neighbourhood-based collaborative filtering' then proceed in two steps:

- The similarity between user i, for which a recommendation shall be made, and user j is calculated.
- The ratings of user j for each item x are weighted with the similarity to calculate a prediction of the rating of user i for item x
- The recommendation is the item with the highest predicted rating (which user i has not bought yet)

In order to do these calculations on encrypted data, we propose the following modifications depicted in Figure 5. These modifications include a (small) pre- and post-processing of the data, which we keep as simple as possible, and modifications of the actual algorithm.

- Most recommender systems of this type operate on sparsly populated rating matrices. However, to hide the information which ratings exist from the data processor the matrices need to be fully populated. This is done in the pre-processing phase, e.g., by completing the matrices with 0 entries.
- The query, which contains the ratings of the user for which a recommendation shall be calculated also need to be 'fully populated' and are, therefore, pre-processed in the same way.

- Since it is not possible to select the item with the highest predicted rating and return this item as recommendation, a predicted rating must be calculated for all items. The maximum then has to be taken during the post-processing phase on the client (data curator)-side.
- Most proposed functions to calculate the similarity between users and to predict the ratings make heavy use of functions such as division (especially for renormalization), square roots or trigonometric functions. We therefore give a modified algorithm to calculate similarity and rating predictions. This can potentially lead to modified recommendations. We analyze this impact in Section 5.

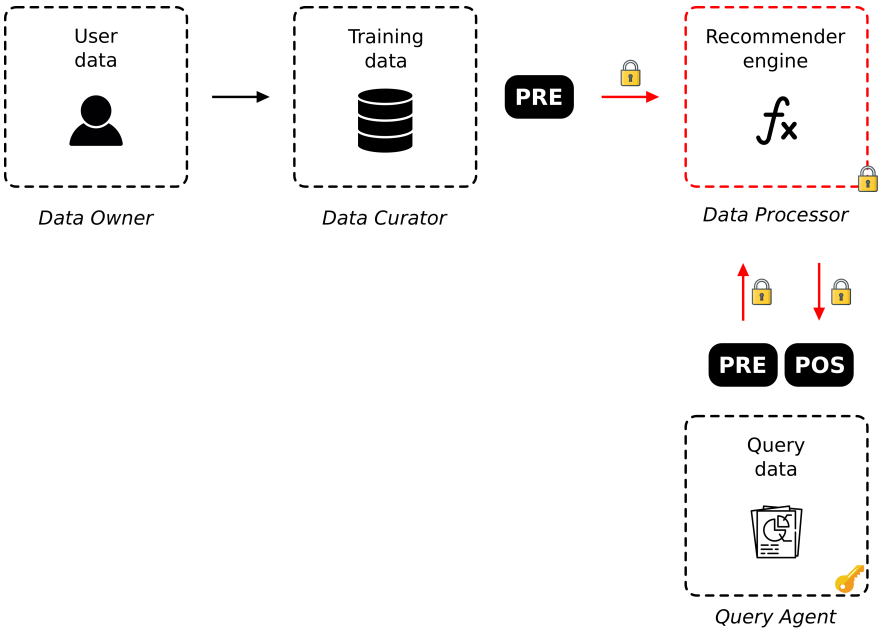

Figure 5: The data need to be pre- and post-processed and the algorithm needs to be modified for the system to be implementable with homomorphic encryption.

Let us now consider a few specific textbook examples of rating matrices and possible modification to allow for a homomorphic calculation of these recommendations.

Unary rating matrix In the case of a unary rating matrix there is only the possibility to express a 'like' of user j for item x by $u_{jx} = 1$. E.g., a 1 indicates an item the user has bought. We first need to complete the matrix, i.e., we will denote all empty entries by 0.

The predicted rating shall be $\tilde{u}_{ix} = \frac{\sum_j sim(i,j) \cdot u_{jx}}{\sum_j |sim(i,j)|}$ where $sim(i,j) = \frac{\sum_x u_{ix} \cdot u_{jx}}{\sqrt{\sum_x u_{ix}^2}\sqrt{\sum_x u_{jx}^2}}$

We propose the following modified prediction to make the calculations on homomorphically encrypted data:

- the similarity shall be calculated as
  - $sim(i,j) = \sum_x u_{ix} \cdot u_{jx}$ (indicating simply the number of items both users have bought), or

– $sim(i,j) = \sum_x u_{ix} \cdot u_{jx} + \sum_x (u_{ix} - 1) \cdot (u_{jx} - 1)$

- The prediction shall be $\tilde{u}_{ix} = \sum_j sim(i,j) \cdot u_{jx}$ (this factor is not normalized, but this is irrelevant, since we will anyway take the highest item)
- return the vector $(u_{ix} - 1) \cdot (u_{ix} - 1) \cdot \tilde{u}_{ix}$ (this eliminates items the user has already bought since it is multiplied by 0). Now the client simply has to recommend the item with the highest entry.

The first way to calculate the similarity obviously has the disadvantage that users who have bought very many items are counted too much, while the second will probably not make accurate predictions for users who have not made many purchases.

Binary rating matrix In the case of a binary rating matrix, the user can express 'likes' (1) and 'dislikes' (-1). We again need to complete the matrix, i.e., we will denote all empty entries by 0.

We propose the following in a homomorphic way:

- the similarity shall be calculated as $sim(i,j) = \sum_x u_{ix} \cdot u_{jx}$ (note that this can be negative)
- The prediction shall be $\tilde{u}_{ix} = \sum_j sim(i,j) \cdot u_{jx}$
- return the vector $(u_{ix} \cdot u_{ix} - 1) \cdot (u_{ix} \cdot u_{ix} - 1) \cdot \tilde{u}_{ix}$ (this eliminates items the user has already rated). Now the client simply has to recommend the item with the highest entry.

Interval-based rating matrix In an interval-based rating matrix, the user rate items on a certain range between 1 and n, i.e., 1 to 7 (the higher the rating, the more the user likes the item). We again need to complete the matrix, i.e., we will denote all empty entries by 0.

The way often used to calculate a prediction in this case is by taking the similarity to be the inverse of the Euclidian distance between user i and j. This is not possible, since we can neither calculate the square root, nor the inverse.

We propose the following in a homomorphic way:

- the similarity shall be calculated as $sim(i,j) = \|\sharp Items\| \cdot n - \sum_x (u_{ix} - u_{jx})^2$ (the first term being the maximal Euclidian distance possible)
- The prediction shall be $\tilde{u}_{ix} = \sum_j sim(i,j) \cdot u_{jx}$

In this case, we cannot simply eliminate the items which the user has already bought. We recommend doing this calculation also on the client side or, alternatively, for the client to additionally provide a 'mask' indicating which items user i has already bought.

5 Experimental Results

Consider the case in which we have 128 products and 100 users with unary ratings. 1 representing that users have a specific financial product in his/her portfolio and 0 saying that the user never traded that financial product.

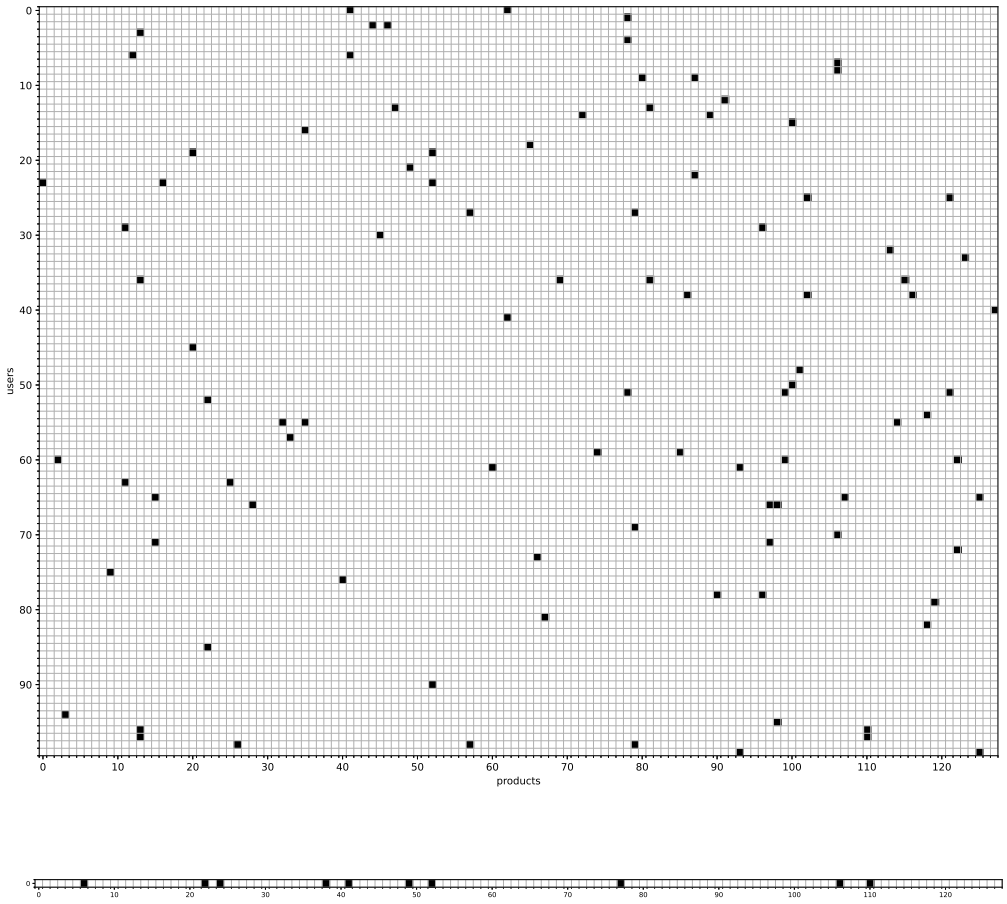

Figure 6: Example of unary rating matrix used for training the model and the unary vector sent by the query agent

Our goal is to be able to say to the query agent that a user who bought product A also bought product B, similar to Amazon recommendations when a user adds to the cart or purchases a new item.

We are well aware that the value of the ratings calculated through the usual recommender system and that performed in the encrypted data will be different. On the former we normalize the similarity and the rating whereas on the latter we proceed without any kind of normalization due to the limitations of the homomorphic encryption scheme (no divisions). The ordering of the recommendations should be similar though. In this simple example it is evident that the ordering is not exaclty the same, but similar. Furthermore, whereas the usual recommender returns a value almost instaneously, due to the low dimensions of the matrix, the homomorphic scheme took about 35 minutes to return a value.

Table 2: Ordering of the recommendations

order	Homorphic		Usual	
	product	rating	product	rating
1	106	3	106	0.26995692
2	52	3	52	0.20556833
3	41	2	22	0.17997128
4	22	2	41	0.12725891
5	110	2	110	0.12725891
6	13	2	13	0.12725891
7	0	1	49	0.08998564
8	12	1	12	0.06362946
9	62	1	62	0.06362946
10	20	1	20	0.06362946
11	49	1	0	0.05195323
12	16	1	16	0.05195323

6 Conclusion and Next Steps

It is possible, but not yet really practical, since it is computationally extremely expensive.

Recommender systems usually take advantage of the sparsity of the matrices involved. In an homomorphically encrypted scheme, all entries need to be encrypted (since it must not be possible to gain knowledge about which entries are non-existant). This blows up the problem.

This is even more of a problem since calculations in homomorphic encryption are extremely slow. The parameters of the encryption scheme must therefore be highly optimized depending on the exact problem (e.g. size of the input rating matrix).

References

[1] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 868–886, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[3] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[4] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[5] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[6] Craig Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, March 2010.

[7] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[8] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptography – ICISC 2012, International Conference on Information Security and Cryptology - ICISC 2012, Lecture Notes in Computer Science, to appear*. Springer Verlag, December 2012.

[9] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1209–1222, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Oliver Masters, Hamish Hunt, Enrico Steffinlongo, Jack Crawford, Flavio Bergamaschi, Maria E. Dela Rosa, Caio C. Quini, Camila T. Alves, Feranda de Souza, and Deise G. Ferreira. Towards a homomorphic machine learning big data pipeline for the financial services sector. Cryptology ePrint Archive, Report 2019/1113, 2019. `https://eprint.iacr.org/2019/1113`.

[11] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 26(1):96–99, 1983.

[12] Microsoft SEAL (release 3.5). `https://github.com/Microsoft/SEAL`, April 2020. Microsoft Research, Redmond, WA.