Privacy-preserving weighted Slope One predictor for Item-based Collaborative Filtering

Anirban Basu[1], Hiroaki Kikuchi[1], and Jaideep Vaidya[2]

[1] Graduate School of Engineering, Tokai University,
1117, Kitakaname, Hiratsuka, Kanagawa, 259-1292, Japan
kikn@tokai.ac.jp, abasu@cs.dm.u-tokai.ac.jp
[2] MSIS Department, Rutgers, The State University of New Jersey
1, Washington Park, Newark, New Jersey, 07102-1897, USA
jsvaidya@business.rutgers.edu
**

Abstract. Rating-based collaborative filtering (CF) predicts the rating that a user will give to an item, derived from the ratings of other items given by other users. Such CF schemes utilise either user neighbourhoods (i.e. user-based CF) or item neighbourhoods (i.e. item-based CF). Lemire and MacLachlan [1] proposed three related schemes for an item-based CF with predictors of the form $f(x) = x + b$, hence the name "slope one". Slope One predictors have been shown to be accurate on large datasets. They also have several other desirable properties such as being updatable on the fly, efficient to compute, and work even with sparse input. In this paper, we present a privacy-preserving item-based CF scheme through the use of an additively homomorphic public-key cryptosystem on the weighted Slope One predictor; and show its applicability on both horizontal and vertical partitions. We present an evaluation of our proposed scheme in terms of communication and computation complexity.

1 Introduction

The proliferation of services provided over the World Wide Web has resulted in individuals having to sift through overwhelming volumes of information (e.g. from social networks, e-commerce catalogs, amongst others), and are thus accosted with the problem of information overload [2]. More recently, services offered through cloud computing have only compounded the problem. This is where *recommendation systems* have come to the rescue.

Most automated recommendation systems employ two techniques: *profile-based* and *collaborative filtering* (CF). The former involves relevant details about users (i.e. information that relate to their tastes) are collected in order to match the items to be recommended to them. In contrast, prediction through CF results from the recorded preferences of the community. While profile-based recommendation for a user with rich profile information can be thorough, CF is fairly accurate within reason, without the need for the user's preferential history. CF has, thus, positioned itself as one of the predominant means of generating recommendations.

Based on filtering techniques, CF is broadly classified into: *memory-based* or *neighbourhood-based* and *model-based*. In *memory-based* approaches, recommendations are developed from user

** The work at Tokai University has been supported by the Japanese Ministry of Internal Affairs and Communications funded project "Research and Development of Security Technologies for Cloud Computing" involving Tokai University, Waseda University, NEC, Hitachi and KDDI. Jaideep Vaidya's work is supported in part by the United States National Science Foundation under Grant No. CNS-0746943 and by the Trustees Research Fellowship Program at Rutgers, The State University of New Jersey.

or item neighbourhoods, based on some sort of proximity (or deviation) measures between opinions of the users, or the ratings of the items, e.g. cosine similarity, Euclidean distance and various statistical correlation coefficients. Memory-based CF can also be distinguished into: *user-based* and *item-based*. In the former, CF is performed using neighbourhood between users computed from the ratings provided by the different users. The latter is item-based where prediction is obtained using item neighbourhoods, i.e. proximity (or deviation) of ratings between various items.

Model-based approaches, in contrast, are sometimes more applicable on large datasets for which some memory-based approaches do not scale well. In model-based approaches, the original user-item ratings dataset is used to *train* a compact model, which is then used for prediction. The model is developed by methods borrowed from artificial intelligence, such as Bayesian classification, latent classes and neural networks; or, from linear algebra, e.g. singular value decomposition (SVD), latent semantic indexing (LSI) and principal component analysis (PCA). Model-based algorithms are usually fast to query but relatively slow to update.

Obviously, CF based approaches perform better with the availability of more data. Furthermore, it may be possible to perform cross domain recommendations, if the corresponding data can be utilised (e.g. a person with a strong interest in horror movies may also rate certain Halloween products highly). However, sharing user-item preferential data for use in CF poses significant privacy and security challenges. Competing organisations, e.g. Netflix and Blockbuster may not wish to share specific user information, even though both may benefit from such sharing. Users themselves might not want detailed information about their ratings and buying habits known to any single organisation. To overcome this, there has been recent work in privacy-preserving collaborative filtering (PPCF) that can enable CF without leaking private information. Indeed, in CF, achieving accuracy and preserving privacy are orthogonal problems. The two main directions of research in privacy-preserving collaborative filtering are: *encryption-based* and *randomisation-based*. In encryption-based techniques, prior to sharing individual user-item ratings data are encrypted using cryptosystems that support homomorphic properties. In randomisation-based privacy preserving techniques, the ratings data is randomised either through random data swapping or data perturbation or anonymisation. We believe that the existing schemes are either impractical from the efficiency standpoint (especially, from the perspective of updating) or from the security standpoint. To improve on this, in this paper, we propose a privacy-preserving (with encryption strategy) item-based collaborative filtering scheme extended from the well-known weighted Slope One predictor [1].

1.1 Our contribution

The contributions of this paper can be summarised as follows: (i) ours is the first work of its kind where a privacy-preserving collaborative filtering scheme is presented using the weighted Slope One predictor and applied to both horizontal and vertical partitions; (ii) our proposal retains the high level of accuracy of Slope One predictors while also providing a high level of privacy – since we are using encryption and security primitives, we can prove that our protocols do not leak any information other than the result. However, due to lack of space, we defer the formal proof of privacy for now; (iii) our proposal has relatively low computation and communication complexity.

The rest of the paper is organised as follows: we briefly present background and overview of preliminaries including the key related work in this area in §2. We summarise our problem statements in §3. In §4, we propose a PPCF scheme based on the weighted Slope One predictor and apply it on horizontal and vertical partitions. In §5, we present evaluations and implementation results followed by a conclusion and promising future directions in §6.

2 Background and Preliminaries

2.1 The weighted Slope One predictor

Lemire and MacLachlan proposed [1] a CF scheme based on predictors of the form $f(x) = x + b$, hence the name "slope one". However, to our knowledge, no one has applied privacy preserving techniques on slope one based CF. Before delving further into PPCF, we present a brief overview of the Slope One predictors. Conforming with realistic datasets, in the following example, we will use the discrete integral range of ratings $[1-5]$ with "0" or "-" or "?" representing absence of ratings. Table 2.1 shows a simple user-item ratings matrix of users rating airlines companies.

The simplest Slope One prediction of rating for any user for an item i_1 given the user's rating for i_2 (i.e. r_{i_2}), is of the form $r_{i_1} = \overline{\delta_{i_1,i_2}} + r_{i_2}$ where $\overline{\delta_{i_1,i_2}}$ is the average deviation of the ratings of item i_1 from those of item i_2 while r_{i_2} is the rating the user has given to item i_2. The average deviation of ratings between a pair of items is calculated using only those ratings where both items have been rated by the same user.

Table 2.1. A simple three users, three items rating matrix.

	British Airways	Emirates	Cathay Pacific
Kikuchi Hiroaki	2	4	4
Jaideep Vaidya	2	5	4
Basu Anirban	1	?	4

Using the *unweighted* Slope One predictor, we derive the missing rating as:

$$? = \frac{(\frac{(4-2)+(5-2)}{2} + 1) + (\frac{(4-4)+(5-4)}{2} + 4)}{2} = 4.0$$

The unweighted scheme estimates a missing rating using the average deviation of ratings between pairs of items with respect to their *cardinalities*. Slope One CF can be evaluated in two stages: pre-computation and prediction of ratings. In the pre-computation stage, the average deviations of ratings from item a to item b is given as:

$$\overline{\delta_{a,b}} = \frac{\Delta_{a,b}}{\phi_{a,b}} = \frac{\sum_i \delta_{i,a,b}}{\phi_{a,b}} = \frac{\sum_i (r_{i,a} - r_{i,b})}{\phi_{a,b}} \tag{2.1}$$

where $\phi_{a,b}$ is the count of the users who have rated both items while $\delta_{i,a,b} = r_{i,a} - r_{i,b}$ is the deviation of the rating of item a from that of item b both given by user i.

In the prediction stage, the rating for user u and item x using the *weighted* Slope One is predicted as:

$$r_{u,x} = \frac{\sum_{a|a\neq x} (\overline{\delta_{x,a}} + r_{u,a})\phi_{x,a}}{\sum_{a|a\neq x} \phi_{x,a}} = \frac{\sum_{a|a\neq x} (\Delta_{x,a} + r_{u,a}\phi_{x,a})}{\sum_{a|a\neq x} \phi_{x,a}}. \tag{2.2}$$

Thus, we can precompute the *difference (or deviation) matrix*[3] $\Delta = \{\Delta_{a,b}\}$ and the *cardinality matrix* $\phi = \{\phi_{a,b}\}$. Note that for space efficiency, we only need to calculate the upper triangulars of those matrices because the lower triangulars can be easily derived from the upper ones, and the leading diagonals are irrelevant. The weighted Slope One has been found to be efficient, e.g.

[3] Note that we do not need to compute average differences according to equation 2.2.

achieving a mean absolute error (MAE) rate close to 0.7 on the MovieLens 100K dataset[4], which is better than CF schemes using cosine similarity or the Singular Value Decomposition, using a reference implementation in Apache Mahout[5].

2.2 Related work

There are a number of existing works on privacy-preserving collaborative filtering (PPCF). One of the earliest such efforts is due to [3] which uses a partial Singular Value Decomposition (SVD) model and homomorphic encryption to devise a multi-party PPCF scheme. In [4], the authors propose a naïve Bayesian classifier based CF over a P2P topology where the users protect the privacy of their data using masking, which is comparable to randomisation. Another homomorphic encryption based SVD scheme has been proposed in [5] but the authors also describe that their scheme does not scale well for realistic datasets; while a randomisation based SVD approach is described in [6]. A general survey of privacy preserving data mining in presented in [7].

2.3 An additively homomorphic public-key cryptosystem – Paillier

Paillier introduced a public-key cryptosystem [8] with additively homomorphic properties. Denoting encryption and decryption functions as $\mathcal{E}()$ and $\mathcal{D}()$ respectively, the encryption of the sum of two plaintext messages m_1 and m_2 is the modular product of their individual ciphertexts:

$$\mathcal{E}(m_1 + m_2) = \mathcal{E}(m_1) \cdot \mathcal{E}(m_2) \tag{2.3}$$

and, the encryption of the product of one plaintext messages m_1 and a plaintext integer multiplicand π is the modular exponentiation of the ciphertext of m_1 with π as the exponent:

$$\mathcal{E}(m_1 \cdot \pi) = \mathcal{E}(m_1)^{\pi}. \tag{2.4}$$

Paillier cryptosystem in described in three steps: *key generation* (algorithm 2.1), *encryption* (algorithm 2.2) and *decryption* (algorithm 2.3).

Algorithm 2.1 Paillier cryptosystem key generation.

1: Generate two large prime numbers p and q, each with half the specified modulus bit length for the cryptosystem.

Ensure: $gcd(pq, (p-1)(q-1)) = 1$ and $p \neq q$.

2: Modulus $n = pq$.

3: Pre-compute n^2.

4: Compute $\lambda = lcm(p-1, q-1) = \frac{(p-1)(q-1)}{gcd(p-1,q-1)}$.

5: $g \leftarrow (1+n)$. {Optimised here but originally: select random $g \in Z^*_{n^2}$ such that n divides the order of g.}

Ensure: $gcd(L(g^{\lambda} \mod n^2), n) = 1$ where $L(u) = \frac{u-1}{n}$. {Optimisation: $g^{\lambda} \mod n^2 = (1 + n\lambda) \mod n^2$.}

6: Pre-compute the modular multiplicative inverse $\mu = L(g^{\lambda} \mod n^2)^{-1} \mod n$.

7: **return** Public key: (n, n^2, g) and Private key: (λ, μ).

[4] `http://www.grouplens.org/node/73`

[5] `http://mahout.apache.org/`

Algorithm 2.2 Paillier encryption algorithm.

Require: Plaintext $m \in Z_n$.

1: Choose random $r \in Z_n^*$.

2: **return** Ciphertext $c \leftarrow (1+mn)\, r^n \mod n^2$. {Optimised here but originally: $c \leftarrow g^m r^n \mod n^2$.}

Algorithm 2.3 Paillier decryption algorithm.

Require: Ciphertext $c \in Z_{n^2}^*$.

1: **return** Plaintext $m \leftarrow L(c^\lambda \mod n^2)\mu \mod n$.

Generalised threshold variant – the Damgärd-Jurik cryptosystem The Damgärd-Jurik cryptosystem [9] proposes a generalisation and a threshold variant of the Paillier cryptosystem using modulo n^{s+1} computations for any natural number $s \geq 1$ (with $s = 1$ for Paillier). It allows for private key sharing between k parties. The ciphertext can be decrypted only after combining all k partial decryptions. In terms of performance, it is slightly slower than Paillier. Security of this cryptosystem is the same as that of the Paillier cryptosystem based on the *decisional composite residuosity assumption*. Note that in our proposed scheme, we assume a *semi-honest* model for the participating sites. Hence, we do not require Damgärd-Jurik zero-knowledge proofs (ZKPs) for the various cryptographic operations from the participating sites.

3 Problem statement

We first define our problem statement in generic terms, and then make it more specific.

Definition 1 (Privacy-Preserving Slope One Predictors). *Given a dataset consisting of m users $u_1, \ldots, u_m$ and n items $it_1, \ldots, it_n$ distributed between a set of k sites, $\mathcal{S}_0, \ldots, \mathcal{S}_{k-1}$ in some fashion, build the weighted Slope One predictor for each item without leaking the privacy of the data owned by site $\mathcal{S}_i$ to any other site $\mathcal{S}_j$.*

Depending on the situation at hand, the collaborative protocol to build the Slope One predictor might result in all of the sites holding the predictor or only some master site holding it, which is still untrusted, as far as the raw data goes. While data may be arbitrarily partitioned between the sites, in general, it is much more likely that the data is partitioned in a particular fashion in real life. We now discuss two specific partitions – horizontal partitioning and vertical partitioning. We present a specific problem statement for each case.

3.1 Horizontal Partitioning of Data

A perfect horizontal partition of a user-item ratings dataset is such that multiple sites (or organisations) contain completely disjoint sets of users but the same set of items. In figure 3.1, a partition of a multiple user, two-items dataset is shown where the data is present in three sites.

In this case, while each site can independently build the Slope One predictor for each item, the predictors are likely to much more accurate when built over the entire data set.

Definition 2 (Privacy-Preserving Slope One Predictors for Horizontally Partitioned Data). *Given a set of k sites, $\mathcal{S}_0, \ldots, \mathcal{S}_{k-1}$, where each site $\mathcal{S}_i$ owns the data about m_i users $u_{i1}, \ldots, u_{im_i}$, such that $\sum_i m_i = m$ and all sites collect information about the same n items $it_1, \ldots, it_n$, build the weighted Slope One predictor for each item without leaking the privacy of the data owned by site $\mathcal{S}_i$ to any other site $\mathcal{S}_j$.*

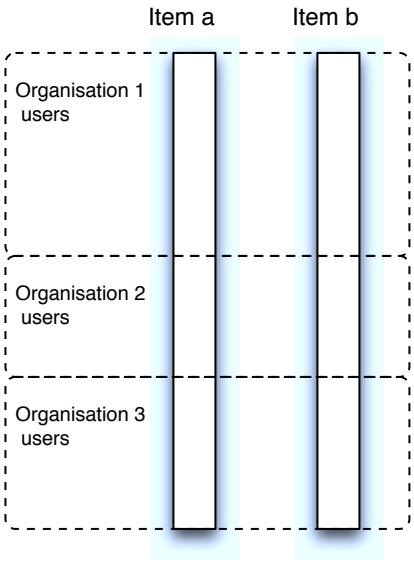

Fig. 3.1. A visualisation of a horizontal partition.

3.2 Vertical Partitioning of Data

A perfect vertical partition is one in which the multiple sites contain the same set of users but completely disjoint sets of items. In figure 3.2, a partition of a multi-user multi-item dataset is shown where there are three item sets across three sites.

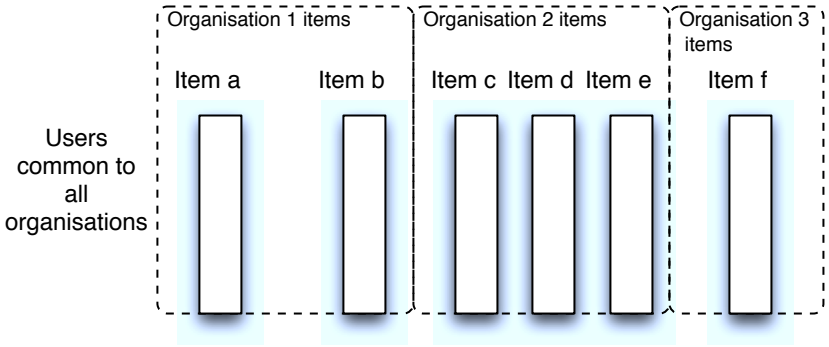

Fig. 3.2. A visualisation of a vertical partition.

In this case, while each party can build a Slope One predictor for its items based on the other items it possesses, the global Slope One predictor built using all of the other items is likely to be significantly more accurate. Therefore, the parties must collaboratively build the predictor.

Definition 3 (Privacy-Preserving Slope One Predictors for Vertically Partitioned Data). *Given a set of k sites, $\mathcal{S}_0, \ldots, \mathcal{S}_{k-1}$, where each site $\mathcal{S}_i$ owns the data about n_i items $it_{i1}, \ldots, it_{nm_i}$, such that $\sum_i n_i = n$ and all sites collect information about the same m users $u_1, \ldots, u_m$, build the Slope One predictor for each item without leaking the privacy of the data owned by site $\mathcal{S}_i$ to any other site $\mathcal{S}_j$.*

Note that in both cases, since the Slope One predictor can be easily built (see equation 2.2) given the deviation matrix and the cardinality matrix, we limit our attention to computing these two matrices in a privacy-preserving fashion.

4 Privacy-preserving Slope One

Below, we show that prediction of a rating, given in equation 2.2, can be derived from encrypted deviations and plaintext cardinalities. Encrypted deviations can be completely decrypted by combining the partial decryptions sites by k sites using the Damgård-Jurik cryptosystem[6].

$$\sum_{a|a\neq x} (\Delta_{x,a} + r_{u,a}\phi_{x,a}) = \mathcal{D}(\prod_{a|a\neq x} (\mathcal{E}(\Delta_{x,a})(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))) \tag{4.1}$$

$$\implies r_{u,x} = \frac{\mathcal{D}(\prod_{a|a\neq x}(\mathcal{E}(\Delta_{x,a})(\mathcal{E}(r_{u,a})^{\phi_{x,a}})))}{\sum_{a|a\neq x}\phi_{x,a}}. \tag{4.2}$$

We now look at how to compute the deviation (Δ) and cardinality (ϕ) matrices when the data is distributed. Note that although local cardinalities at each site are encrypted, the global cardinalities should be decrypted to fit into the prediction equation 4.2. Also, the prediction equation contains a number of $\mathcal{E}(r_{u,a})$ values, which are provided encrypted in a query to obtain the prediction.

4.1 Horizontal

If we denote the deviation matrix for all item pairs as Δ and the cardinality matrix as ϕ, and denote organisational sites as $\mathcal{S}_{i|i=1\ldots n}$ then site $\mathcal{S}_i$ owns $\Delta^{\mathcal{S}_i}$ and $\phi^{\mathcal{S}_i}$ respectively. An element in the deviation matrix between items "a" and "b" is given as $\Delta^{\mathcal{S}_i}_{a,b}$ owned by $\mathcal{S}_i$, if any. Since the user sets are completely disjoint between the organisations, the following two equations can be easily inferred:

$$\Delta = \sum_i \Delta^{\mathcal{S}_i} \implies \text{any } \Delta_{a,b} = \mathcal{D}(\prod_i \mathcal{E}(\Delta^{\mathcal{S}_i}_{a,b})) \text{ and} \tag{4.3}$$

$$\phi = \sum_i \phi^{\mathcal{S}_i} \implies \text{any } \phi_{a,b} = \mathcal{D}(\prod_i \mathcal{E}(\phi^{\mathcal{S}_i}_{a,b})). \tag{4.4}$$

Initial matrix aggregation Each site independently calculates its own deviation and cardinality matrices ranging over all pairs of items from the user set it has[7]. To aggregate the matrices, any site $\mathcal{S}_i$ should start a k-cycle (k being the number of collaborating sites) by sending its neighbour, $\mathcal{S}_{i+1}$, the upper triangulars of $\mathcal{E}(\Delta^{\mathcal{S}_i})$ and $\mathcal{E}(\phi^{\mathcal{S}_i})$. Site $\mathcal{S}_{i+1}$ can then aggregate these with its own deviation and its own cardinality matrices by homomorphic addition and pass the results on to its neighbour $\mathcal{S}_{i+2}$. Once the cycle finishes, i.e. the encrypted matrices reach $\mathcal{S}_i$ in a cycle, it can then forward the updated final encrypted matrices in one more k-cycle so that its neighbours can all maintain the same up-to-date matrix. This process is described in algorithm 4.1.

[6] In the Paillier and the Damgård-Jurik cryptosystems, encryption of a negative integer can be calculated by using its modular additive inverse, i.e. $E(-x) = E(n-x)$, so we can treat all $x > \frac{n}{2}$ to be negative.

[7] Recall the need to calculate only upper triangulars.

Algorithm 4.1 Initial matrix aggregation in the horizontal partitioning scheme.

Require: Each site $\mathcal{S}_i$ independently calculates its own deviation matrix $\Delta^{\mathcal{S}_i}$ and its own cardinality matrix $\phi^{\mathcal{S}_i}$.

1: **for** each site $\mathcal{S}_i$ in the k sites **do**
2: $\mathcal{S}_i$ sends $\mathcal{E}(\Delta^{\mathcal{S}_i})$ and $\mathcal{E}(\phi^{\mathcal{S}_i})$ to its neighbour $\mathcal{S}_{i+1}$.
3: $\mathcal{S}_{i+1}$ computes homomorphic addition: $\mathcal{E}(\Delta^{\mathcal{S}_i})\mathcal{E}(\Delta^{\mathcal{S}_{i+1}})$ and $\mathcal{E}(\phi^{\mathcal{S}_i})\mathcal{E}(\phi^{\mathcal{S}_{i+1}})$; then sends this to its neighbour $\mathcal{S}_{i+2}$ until all sites are reached in this k-cycle.
4: **end for**
5: {In k such forwarding operations, the complete encrypted matrices $\mathcal{E}(\Delta) = \prod_i \mathcal{E}(\Delta^{\mathcal{S}_i})$ and $\mathcal{E}(\phi) = \prod_i \mathcal{E}(\phi^{\mathcal{S}_i})$ will reach $\mathcal{S}_i$.}
6: **for** each site $\mathcal{S}_i$ in the k sites **do**
7: $\mathcal{S}_i$ forwards $\mathcal{E}(\Delta)$ and $\mathcal{E}(\phi)$ to $\mathcal{S}_{i+1}$ until all sites are reached.
8: **end for**
9: {In this cycle, every $\mathcal{S}_i$ gets $\mathcal{E}(\Delta)$ and $\mathcal{E}(\phi)$, where $\mathcal{E}(\Delta) = \mathcal{E}(\sum_i \Delta^{\mathcal{S}_i})$ and $\mathcal{E}(\phi) = \mathcal{E}(\sum_i \phi^{\mathcal{S}_i})$.}

Matrix update Any changes in individual ratings will change the individual pairwise deviations (and also cardinalities if new users are added or old users are removed), which will trigger an update in a cycle akin to the matrix aggregation process. If $\Delta_{a,b}$ is changed to $\Delta'_{a,b}$ at site $\mathcal{S}_i$ then it can pass it on encrypted to its neighbour $\mathcal{S}_{i+1}$ in a k-cycle (k being the number of collaborating servers), which in turn can pass it on to its neighbour, $\mathcal{S}_{i+2}$ so that each site can update its $\mathcal{E}(\Delta_{a,b})$ to $\mathcal{E}(\Delta'_{a,b})$. The cardinality matrix elements can be updated in the same way. This update process is described in algorithm 4.2.

Algorithm 4.2 Matrix update in the horizontal partitioning scheme.

Require: $\Delta_{a,b}$ is changed to $\Delta'_{a,b}$ at site $\mathcal{S}_i$ because of change of ratings for item "a" or item "b" or additional or removal of users at site $\mathcal{S}_i$. Alternatively, ϕ may have changed to ϕ' at site $\mathcal{S}_i$.

1: **for** each site $\mathcal{S}_i$ in the k sites **do**
2: If the update is required, $\mathcal{S}_i$ forwards $\mathcal{E}(\Delta'_{a,b})$ to $\mathcal{S}_{i+1}$ until all sites are reached.
3: If the update is required, $\mathcal{S}_i$ forwards $\mathcal{E}(\phi'_{a,b})$ to $\mathcal{S}_{i+1}$ until all sites are reached.
4: **end for**

We will have to decrypt the global cardinality matrix $\phi = \mathcal{D}(\prod_i \mathcal{E}(\phi^{\mathcal{S}_i})) = \sum_i \phi^{\mathcal{S}_i}$ before we can apply the prediction equation 4.2 but we can also use the secure sum protocol [10] to add up the local cardinality, i.e. $\phi^{\mathcal{S}_i}$ matrices without encrypting, which will reduce the computation cost but increase the communication cost.

4.2 Vertical

Vertical partitions are not trivial as this involves expansions of the individual servers' deviation and cardinality matrices. This will involve sharing of ratings for each pair of items in which one item (in the pair) exists in one partition and the other item exists in another partition. Clearly, if both items are completely held by any one server, it is easy to calculate the average deviation. However, when both items are held by different servers, a privacy-preserving protocol is needed to compute the average deviation. This can be done as follows. Assume site $\mathcal{S}_x$ holds item a and site $\mathcal{S}_y$ holds item b. Now, $\mathcal{S}_x$ can compute $s_a = \sum_i r_{i,a}$ for all its users and $\mathcal{S}_y$ can compute $s_b = \sum_i r_{i,b}$. Note that $s_a - s_b = \Delta_{a,b} + overcount_a - overcount_b$, where $overcount_a = \sum_i r_{i,a}$, where $r_{i,b}$ does not exist. Similarly, $overcount_b = \sum_i r_{i,b}$, where $r_{i,a}$ does not exist. Since, s_a, s_b

can be computed locally, given a way to compute $overcount_a$, $overcount_b$, $\Delta_{a,b}$ can be computed in a privacy-preserving fashion.

We now show how to compute $overcount_a$, $overcount_b$. Essentially, this can be done using a secure scalar product protocol. Basically, $\mathcal{S}_x$ forms the vector $\boldsymbol{xa}$, ($xa_i = r_{i,a}$, if $r_{i,a}$ exists, otherwise 0). Correspondingly $\mathcal{S}_y$ forms the boolean vector $\boldsymbol{ya}$, ($ya_i = 1$, if $r_{i,b}$ does *not* exist, otherwise 0). Now, $overcount_a = \boldsymbol{xa} \cdot \boldsymbol{ya}$. This can be seen as follows – the crucial part is the correct formulation of the boolean vector. Note that the boolean vector is essentially the inverse of the existence vector for item b. Thus, any rating of a without a corresponding rating of b will of necessity be now added in to the scalar product. This is exactly $overcount_a$.

Similarly, $\mathcal{S}_x$ creates the boolean vector $\boldsymbol{xb}$, ($xb_i = 1$, if $r_{i,a}$ does *not* exist, otherwise 0), while $\mathcal{S}_y$ forms the vector $\boldsymbol{yb}$, ($yb_i = r_{i,b}$, if $r_{i,b}$ exists, otherwise 0). Now, $overcount_b = \boldsymbol{xb} \cdot \boldsymbol{yb}$. The reasoning is exactly as before – any rating of b without a corresponding rating of a will now be added into the scalar product.

We can easily use the additively homomorphic cryptosystem to do the scalar product. Basically, $\mathcal{S}_x$ can encrypt $\boldsymbol{xa}$ and send it to $\mathcal{S}_y$ who can multiply the correct encryptions together to get the encrypted form of $overcount_a$. Similarly, $\mathcal{S}_y$ can encrypt $\boldsymbol{yb}$ and send it to $\mathcal{S}_x$ who can multiply the correct encryptions together to get the encrypted form of $overcount_b$. Deriving $\phi_{a,b}$ is similarly easy – both $\mathcal{S}_x$ and $\mathcal{S}_y$ simply encode the existence vectors for items a and b respectively. The scalar product of these two vectors is $\phi_{a,b}$. Note that typically, the secure scalar product is computed in split fashion. i.e., for $overcount_a$ the two sites get o_x and o_y respectively such that $o_x + o_y = overcount_a$. This is to ensure that the scalar product itself does not directly reveal any information. This works well for us since, when computing $\Delta_{a,b}$ both sites can locally sum up their shares and finally sum up the whole in encrypted form. For the sake of simplicity, in the algorithm, we just note that the scalar products are computed securely. The overall matrix aggregation process is described in algorithm 4.3.

Algorithm 4.3 Initial matrix aggregation in the vertical partitioning scheme.

{As before note that only upper triangulars of the deviation and relative occurence matrix needs to be computed as the lower triangulars can be easily deduced from the upper ones. The leading diagonal is irrelevant.}
for each cell (a,b) in the upper triangular matrix **do**
  if both item a and item b are held by the same party $\mathcal{S}_k$ **then**
    $\mathcal{S}_k$ computes $\Delta_{a,b}$ and $\phi_{a,b}$
  else
    Assume site $\mathcal{S}_x$ owns item a while site $\mathcal{S}_y$ owns item b
    $\mathcal{S}_x$ computes $s_a = \sum_i r_{i,a}$
    $\mathcal{S}_y$ computes $s_b = \sum_i r_{i,b}$
    $\mathcal{S}_x$ creates the vector $\boldsymbol{xa}$, ($xa_i = r_{i,a}$, if $r_{i,a}$ exists, otherwise 0)
    $\mathcal{S}_x$ creates the boolean vector $\boldsymbol{xb}$, ($xb_i = 1$, if $r_{i,a}$ does *not* exist, otherwise 0)
    $\mathcal{S}_y$ creates the vector $\boldsymbol{yb}$, ($yb_i = r_{i,b}$, if $r_{i,b}$ exists, otherwise 0)
    $\mathcal{S}_y$ creates the boolean vector $\boldsymbol{ya}$, ($ya_i = 1$, if $r_{i,b}$ does *not* exist, otherwise 0)
    $\mathcal{S}_x$ and $\mathcal{S}_y$ securely compute $overcount_a = \boldsymbol{xa} \cdot \boldsymbol{ya}$
    $\mathcal{S}_x$ and $\mathcal{S}_y$ securely compute $overcount_b = \boldsymbol{xb} \cdot \boldsymbol{yb}$
    $\Delta_{a,b} = s_a - s_b - overcount_a + overcount_b$
    $\mathcal{S}_x$ creates the boolean vector $\boldsymbol{va}$, ($va_i = 1$, if $r_{i,a}$ exists, otherwise 0)
    $\mathcal{S}_y$ creates the boolean vector $\boldsymbol{vb}$, ($vb_i = 1$, if $r_{i,b}$ exists, otherwise 0)
    $\mathcal{S}_x$ and $\mathcal{S}_y$ securely compute $\phi_{a,b} = \boldsymbol{va} \cdot \boldsymbol{vb}$
  end if
end for

Matrix update As earlier, any changes in individual ratings will change the individual pairwise deviations (and also cardinalities if new users are added or old users are removed[8]). The addition of new users is easy to handle. We simply need to carry out the prior aggregration process only for the added new users – these can then be added in encrypted form to the existing $\Delta_{a,b}$ and $\phi_{a,b}$. Similarly, for deleted users, we can carry out the prior aggregation process only for the deleted users and then subtract those from the existing $\Delta_{a,b}$ and $\phi_{a,b}$. The case for updated users is a little more complicated. However, a simple solution is to treat an update as a deletion of the old user and addition of a new user with the updated values. Now, the prior process can be used to perform the updates. This update process is described in algorithm 4.4.

Algorithm 4.4 Matrix update in the vertical partitioning scheme. Note that the update scheme does not mention the process of updating ϕ but it is similar to that of updating Δ.

1: **for** each cell (a,b) in the upper triangular matrix **do**
2: **if** new users are added **then**
3: Compute $ND_{a,b} = \Delta_{a,b}$ over only the new users using Algorithm 4.3
4: Compute $\Delta'_{a,b} = \Delta_{a,b} + ND_{a,b}$
5: **else if** some users are deleted **then**
6: Compute $ND_{a,b} = \Delta_{a,b}$ over only the deleted users using Algorithm 4.3
7: Compute $\Delta'_{a,b} = \Delta_{a,b} - ND_{a,b}$
8: **else**
9: Compute $NDO_{a,b} = \Delta_{a,b}$ over only the modified users (using original values) using Algorithm 4.3
10: Compute $NDN_{a,b} = \Delta_{a,b}$ over only the modified users (using new values) using Algorithm 4.3
11: Compute $\Delta'_{a,b} = \Delta_{a,b} - NDO_{a,b} + NDN_{a,b}$
12: **end if**
13: **end for**
14: {Note that the process for computing ϕ' from ϕ is exactly the same as above.}

Note that, for the sake of simplicity, the derivation of the Δ matrix is described in plaintext form. However, if step 15 of algorithm 4.3, and steps 4, 7, and 11 of algorithm 4.4 is carried out in encrypted form, we can derive the matrix in encrypted form.

5 Implementation and evaluation

We first analyse the computation and communication complexity of our algorithms, and then present a performance evaluation of the cryptographic primitives.

5.1 Computation and Communication Complexity

In the following, note that, as defined in Definition 1, m represents the global number of users and n represents the global number of items.

Horizontal partition Considering the case of horizontal partitioning, the cost of matrix aggregation in Algorithm 4.1 is significant in comparison with cost of computing the local deviation

[8] Note that an user can only be added or deleted by all of the sites together to maintain perfect vertical partitions. We also leave the case of addition or deletion of items to future work.

matrix and cardinality matrix. First, every site needs to encrypt its own deviation and cardinality matrix. Given that only the upper triangulars are computed, this gives $\sum_{i=0}^{n-1} i = n(n-1)/2$ encryptions for each matrix at each site. With k sites, there are a total of $kn(n-1)/2 = O(kn^2)$ encryptions for each matrix. Each site, excepting the first one, also needs to homomorphically add its own matrix to the running aggregate. Thus $O(kn^2)$ multiplications are also required. Finally, the cardinality matrix is decrypted (unless we are using secure sum), giving $n(n-1)/2$ decryptions. Since the cost of the encryptions and decryptions dominate, the overall computation cost is $O(kn^2)$. Now, let us consider the communication cost. If we assume that each matrix is forwarded in a single message, there are a k messages in the first round (to aggregate), and then another k messages to forward, thus giving a total of $2k$ messages. Finally, for the cardinality matrix, the decryptions require another k messages. Since the upper triangular is transmitted in each message, giving a total of $O(kn^2)$ bits of communication. For updating the matrices (Algorithm 4.2), $\mathcal{S}_i$ forwards the updated element(s) of the deviation matrix and/or the cardinality matrix in a k cycle. Since the updated element in each matrix is simply forwarded as an encrypted message, there is no further cost of encryption in the subsequent sites apart from the first site $\mathcal{S}_i$. If the number of items updated is η then the number of encryptions will be $O(\eta^2)$. The number of messages forwarded is now k (because of one cycle), also requiring $O(k\eta^2)$ bits of communication. We will still have to decrypt the entire cardinality matrix, hence another $O(kn^2)$ bits of communication are involved.

Vertical partition Now, consider the vertical partitioning case. This takes significantly larger computation and communication effort. In algorithm 4.3, for every pair of items (a, b) that are held by two different sites, two secure scalar products need to be computed to get the deviation for that cell, while one secure scalar product needs to be computed to get the cardinality. Each scalar product requires m encryptions and m multiplications. Therefore the total cost is $O(m)$. Since there are $n(n-1)/2$ cells, even though all cells do not require secure operations, overall the computation complexity is $O(mn^2)$. Since, typically, $m >> k$, this cost is far larger than the horizontally partitioned data case. Each scalar product requires two rounds of communication, and $O(m)$ bits. Therefore, the overall communication cost is $O(n^2)$ messages, and $O(mn^2)$ bits. In algorithm 4.4, the process is exactly the same as earlier, but only carried out over the changed users. Assuming the number of changed users is given by m', the computation cost is $O(m'n^2)$, and the communication cost is $O(n^2)$ messages and $O(m'n^2)$ bits of communication.

5.2 Performance evaluation of cryptographic primitives

We have implemented the Paillier cryptosystem with homomorphic addition and multiplication functions. The cryptographic primitives of our implementation have been tested on a hardware consisting of a 2.53 GHz Intel Core i5 processor with 8GB DDR3 (1.07GHz bus speed) RAM running 64-bit Mac OS X 10.6.6 and Java 1.6.0_22 64-bit HotSpot Server VM[9]. The results are shown in table 5.1[10]. The results have been generated taking averages over 943 iterations (which happens to be the number of users in the MovieLens 100K dataset).

Given the performance of a Paillier cryptosystem and the aforementioned complexity analysis, one can estimate the time taken for the initial matrix aggregation, matrix updates and queries for realistic datasets, although it is harder to capture communication times in the absence of realistic network conditions. This paper is a work-in-progress and we will present extensive simulation/emulation results in future work.

[9] Arbitrary precision integers are represented using the `java.math.BigInteger` class.

[10] For better accuracy, profiling has been done using `ThreadMXBean::getCurrentThreadUserTime()` method instead of the de-facto `System.nanoTime()`.

Table 5.1. Comparison of a Java implementation of Paillier cryptosystem with different bit lengths for the public key (i.e. modulus n). Plaintext is random and is in $\{1, 2, 3, 4, 5\}$ and integer multiplicand is random and is in $\{0, \ldots, 999\}$ with about 80% of them being zero. Note that the bit length of the plaintext does not have much effect on encryption time due to the optimisation $c = (1 + mn)\, r^n \, mod \, n^2$.

Paillier cryptosystem	**512-bits**	**1024-bits**	**2048-bits**
Average encryption time (ms)	2.656	17.262	124.544
Average decryption time (ms)	2.719	17.127	123.703
Average homomorphic addition time (ms)	0.008	0.027	0.099
Average homomorphic multiplication time (ms)	0.013	0.039	0.132

6 Conclusion and future work

In this paper, we have proposed an efficient privacy-preserving solution for the problem of collaborative filtering over distributed data. Our solution is based on the weighted Slope One predictor from Lemire and MacLachlan [1] and uses homomorphic encryption to carry out the necessary computations. In future, we will present more exhaustive test results as well as comparisons with other CF schemes, which we have left out in this paper due to space constraints. Currently, we assume a PKI infrastructure, and complete collaborative environment. In the future, we are planning to reduce these trust assumptions, by leveraging P2P based topology for the participating sites, and reducing the need for cryptographic operations. One way to do this is to explore the possibility of aggregating partial deviation and cardinality matrix for a subgroup at a higher level aggregator using secure sum based techniques and then propagate it upwards. We will explore this in the future.

References

1. Lemire, D., Maclachlan, A.: Slope one predictors for online rating-based collaborative filtering. Society for Industrial Mathematics (2005)
2. Schafer, J.B., Konstan, J., Riedi, J.: Recommender systems in e-commerce. In: Proceedings of the 1st ACM conference on Electronic Commerce, New York, USA, ACM Press (1999) 158–166
3. Canny, J.: Collaborative filtering with privacy. Proceedings 2002 IEEE Symposium on Security and Privacy (2002) 45–57
4. Kaleli, C., Polat, H.: P2P collaborative filtering with privacy. Turkish Journal of Electric Electrical Engineering and Computer Sciences **8**(1) (2010) 101–116
5. Han, S., Ng, W.K., Yu, P.S.: Privacy-Preserving Singular Value Decomposition. IEEE 25th International Conference on Data Engineering (2009) 1267–1270
6. Polat, H., Du, W.: SVD-based collaborative filtering with privacy. Proceedings of the 20th ACM Symposium on Applied Computing (2005)
7. Aggarwal, C., Yu, P.: 2. In: A General Survey of Privacy-Preserving Data Mining Models and Algorithms. Springer (2008) 11–52
8. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology EUROCRYPT. Volume 1592., Springer (1999) 223–238
9. Damgärd, I., Jurik, M.: A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In: Public Key Cryptography, Springer (2001) 119–136
10. Clifton, C., Kantarcioglu, M., Lin, X., Vaidya, J., Zhu, M.: Tools for Privacy Preserving Distributed Data Mining. SIGKDD Explorations **4**(2) (January 2003) 28–34