Perturbation based privacy preserving Slope One predictors for collaborative filtering

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

[1] Graduate School of Engineering, Tokai University,
2-3-23 Takanawa, Minato-ku, Tokyo 108-8619, Japan
abasu@cs.dm.u-tokai.ac.jp, kikn@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. The prediction of the rating that a user is likely to give to an item, can be derived from the ratings of other items given by other users, through collaborative filtering (CF). However, CF raises concerns about the privacy of the individual user's rating data. To deal with this, several privacy-preserving CF schemes have been proposed. However, they are all limited either in terms of efficiency or privacy when deployed on the cloud. Due to its simplicity, Lemire and MacLachlan's weighted Slope One predictor is very well suited to the cloud. Our key insight is that, the Slope One predictor, being an invertible affine transformation, is robust to certain types of noise. We exploit this fact to propose a random perturbation based privacy preserving collaborative filtering scheme. Our evaluation shows that the proposed scheme is both efficient and preserves privacy.

1 Introduction

Recommender systems have come to the rescue of individuals accosted with the problem of information overload [1] as a result of the numerous services being offered over the World Wide Web, e.g. social networks, e-commerce catalogs, amongst others. Most automated recommendation systems employ two techniques: *profile-based* and *collaborative filtering* (CF). The former puts to use the information that relate to users' tastes 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, 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 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. Such a model can be 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. There are also the fast-to-query, fast-to-update well-known Slope One CF predictors [2].

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.

Due to the privacy concerns, recently there has been significant research on privacy-preserving collaborative filtering (PPCF). The two main classes of solutions are: *encryption-based* and *randomisation-based*. In the encryption-based techniques, prior to sharing individual user-item ratings data are encrypted using cryptosystems that support homomorphic properties. In the randomisation-based privacy preserving techniques, the ratings data is randomised either through random data swapping or data perturbation or anonymisation. Given the large quantities of data, there has been a growing push to perform CF on the cloud[3]. Since CF is typically done at the application level, it is ideally suited to Software as a Service (SaaS) clouds, which are used to deploy applications scalably on the cloud. However, all of the existing PPCF solutions suffer from either scalability or security when deploying on SaaS clouds.

Our key insight is that Lemire and MacLachlan's weighted Slope One predictor is very well suited for SaaS clouds, and being an invertible affine transformation, is robust to certain types of noise. We exploit this fact to propose a random perturbation based privacy preserving collaborative filtering scheme. We now give a short illustrative example to demonstrate this.

1.1 Illustrative example

In (unweighted) Slope One, we usually have $f(x) = x + b$ where x is a rating given in the rating query while b is extracted from the deviation matrix[4]. Let us take a simple example with five ratings of two features X and Y.

```
X = [1 3 1 3 5]
Y = [5 2 4 3 4]
```

If we are to predict Y from X, we can use the basic Slope One predictor as $Y = X + \overline{(Y - X)}$ where the $\overline{(Y - X)}$ (i.e. the mean of the differences between Y and X) can be pre-computed. With the given data, we have $\overline{(Y - X)} = 1$, which gives us our Slope One predictor as a line $Y = X + 1$. If we now added random noise from a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma = 5)$, we have a noisy data as follows.

```
pX = [-0.501160796 4.221286539 6.991751095 -7.917938198 10.47511263]
pY = [-1.388415841 8.382367701 12.66566552 1.829093784 -1.433503247]
```

Given this perturbed data, we have $\overline{(pY - pX)} = 1.357231329$, which gives us our Slope One predictor as a line $pY = pX + 1.357231329$. Thus, the lines represented by $Y = X + 1$ and $pY = pX + 1.357231329$ are parallel with a small offset between them.

[3] For example, Netflix uses Amazon Web Services for their computing needs (`http://techblog.netflix.com/2010/12/four-reasons-we-choose-amazons-cloud-as.html`)

[4] Please see the §2 for the basics of Slope One predictors.

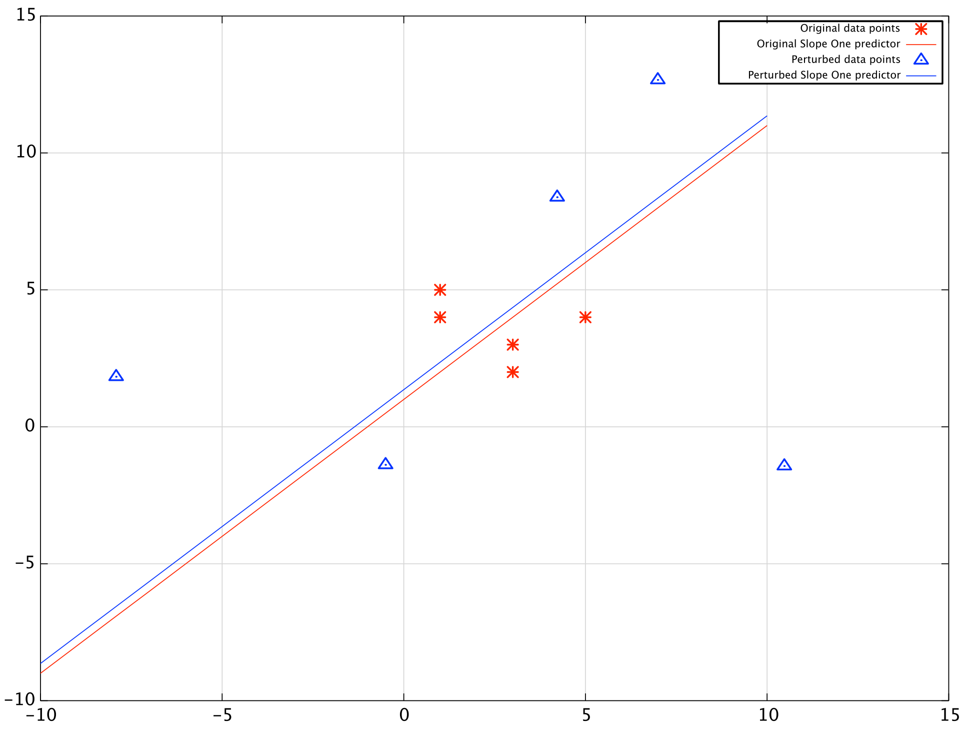


Fig. 1.1. The robustness of Slope One predictors to noise.

In figure 1.1, we plot the scatter digram of the original data points, the perturbed data points and then the two Slope One predictor lines. From the figure, we can infer that despite the the perturbed data being significantly different from the original data, the basis for prediction – the Slope One lines – are very similar in both cases, identifying the robustness of Slope One to additive noise. Thus, the key insight of this paper is that the Slope One predictor is essentially a line on a 2D graph which is largely unaffected by data perturbation and is easy to compute. This lends itself as a robust perturbation based CF suitable for the cloud.

1.2 Our contribution

In this paper, we demonstrate the effect of noise on the weighted Slope One predictor, and then, propose a privacy preserving collaborative filtering scheme. The specific contributions of this paper are: (i) ours is the first attempt, to our knowledge, to have proposed a PPCF scheme using random data perturbation on the weighted Slope One predictor; (ii) we present comparative performance analysis of more than one data perturbation methods alongside other related randomisation-based PPCF work; and (iii) we also take into account implementation concerns in real world cloud computing platforms particularly in terms of scalability.

The rest of the paper is organised as follows: we briefly present background of Slope One in §2 followed the key related work in this area and our previous work in §3. In §4, we examine the effects of random noise on the weighted Slope One predictor leading to the §5 which presents the problem statement in §4.1, proposals of PPCF schemes and a discussion on the level of privacy.

In §6, we present implementation and evaluation results of our proposal followed by a conclusion and promising future directions in §7.

2 Slope One for collaborative filtering

2.1 The weighted Slope One predictor

Lemire and MacLachlan proposed [2] a CF scheme based on predictors of the form $f(x) = x + b$, hence the name "slope one". Before delving into PPCF, we present a brief overview of the Slope One predictors. 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
Alice	2	4	4
Bob	2	5	4
Tracy	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 has two stages: pre-computation (or update) and prediction of ratings. In the pre-computation stage, the average deviations of ratings from item a to item b is given as:

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

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:

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

Thus, we can pre-compute the *difference (or deviation) matrix*[5] $\varDelta = \{\varDelta_{a,b}\}$ and the *cardinality matrix* $\phi = \{\phi_{a,b}\}$, shown in figure 2.1. Note that for space efficiency, we only need

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

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. achieving a mean absolute error (MAE) rate close to 0.7 on the MovieLens 100K dataset[6], which is better than CF schemes using cosine similarity or another CF scheme using the Singular Value Decomposition, using a reference implementation in Apache Mahout[7].

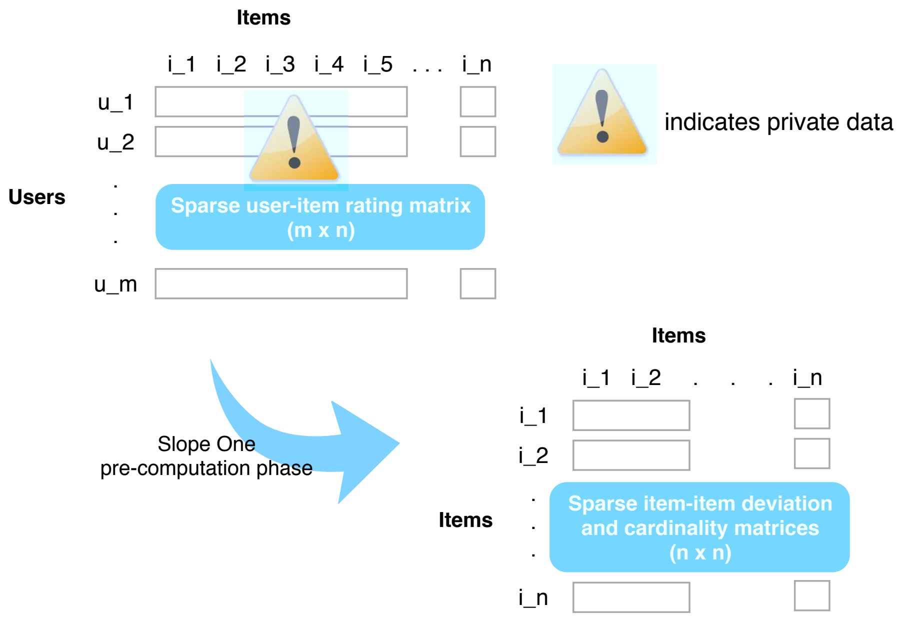

Items
i_1 i_2 i_3 i_4 i_5 . . . i_n
u_1
u_2
Users
Sparse user-item rating matrix (m x n)
u_m
indicates private data
Slope One pre-computation phase
Items
i_1 i_2 . . . i_n
i_1
i_2
Items
Sparse item-item deviation and cardinality matrices (n x n)
i_n

Fig. 2.1. The pre-computation phase of Slope One.

In Slope One, the generalised form $f(x) = x + b$ where the constant b represents the deviation between an item pair makes this predictor particularly immune to additive random noise.

3 Related work

The problem of privacy preservation is compounded by the remarkable scales of real world datasets. Often, in PPCF, achieving efficiency or accuracy on one hand and preserving privacy of user-item preferential data on the other are orthogonal problems. While encryption-based solutions result in no loss of accuracy, homomorphic operations take toll on computational efficiency. In contrast, randomisation-based solutions are many orders of magnitude faster but only at the cost of accuracy. Servers these days are no longer just single units but clusters, or private and public clouds. The design of PPCF solutions ought to consider efficiency (both computational and storage) because of the immediate implications on costs.

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

[6] `http://www.grouplens.org/node/73`
[7] `http://mahout.apache.org/`

(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].

In our recent works [8,9,10], we have proposed PPCF solutions based on the well-known weighted Slope One predictor [2]. Particularly, in [10] we also showed the applicability of our PPCF scheme on a real world public cloud computing platform.

One of the main differences between the existing works (including our own previous works) and the work presented in this paper is that the latter is more *efficient* than solutions using threshold homomorphic encryption while having an *accuracy* comparable to the more accurate encryption based PPCF schemes. Further to that, the model presented in this paper is applicable to a SaaS cloud, which most PPCF solutions are not.

3.1 Types of privacy threats to CF and their solutions

In general, whenever a CF scheme requires intermediate matrices (e.g. Slope One, SVD-based solutions) that do not contain private data, one can either opt for hiding (cryptographically or with perturbation) the values of ratings as they are submitted, or attempt to de-link such submissions from users by using some identity anonymisation mechanism. At query time, depending on how the intermediate matrices have been computed, the user's query vector can also be hidden using similar techniques. The cryptographic procedures ensures no change in accuracy from the original CF scheme but it comes at the cost of performance. The random perturbation methods ensure better performance at the cost of accuracy. In addition, a combination of both may also be usable depending on the CF scheme.

Notice that there is yet another scenario, which we do not cover in this paper. It is a multi-site scenario where user-item rating data is stored unencrypted, unperturbed in each site. The privacy of the data is considered a concern when it is shared amongst cites. This is particularly applicable in cross-domain collaborative filtering where users trust the sites that store their user-item rating data. In this paper, we assume the users do not trust any CF site and therefore the privacy of their data must be preserved throughout.

In figure 2.1, we observe that the user-item rating matrix poses a privacy risk for individual users but the item-item deviation and cardinality matrices do not. This privacy threat exists at the time of pre-computation or update of the deviation and cardinality matrices. There is also a privacy risk at the time of query in which the rating vector of the querying user may be exposed.

Preserving privacy at the time of pre-computation A number of measures can be taken to preserve the privacy at pre-computation time. One well-known procedure is to use a threshold homomorphic cryptosystem. Prior research works [5,9,10] have proposed that over other CF schemes as well as Slope One. Individual user's rating data are encrypted at the time of submission by a shared public key, thus leading to an encrypted deviation matrix. CF queries can then be responded to using the homomorphic property of the cryptosystem and decrypted by trusted third-party threshold decryption servers on behalf of the user. The main downside of this is the dependency on the trusted third party. In addition, the CF computation time is also increased due to the computational overhead of cryptographic operations.

Alternatives to the homomorphic encryption approach include anonymising as well as random perturbation of the data as they are recorded. Depending on the type of CF used, it may be necessary to get rid of the noise added through random perturbation. Techniques for noise removal include Bayesian filtering and spectral filtering amongst others. This is, however, not required in the weighted Slope One for certain types of noise distributions, as we will see in §4. In order to hide the number of ratings a particular user submits, some PPCF schemes normalise the user-item rating data, which essentially converts a sparse data problem into a dense one (e.g. [6]) contributing to significant costs in storage and computation.

Preserving privacy at query time At the time of query, the query vector (containing sensitive ratings) could be encrypted using a homomorphic cryptosystem, thus hiding the private ratings. The result of such queries ought to be decrypted by trusted third party servers. Alternatively, homomorphic public-key encryption may also be used in a different way (e.g. our earlier proposal [10]) whereby the querying user is solely in charge of decrypting the response, which helps eliminating the need for trusted third parties. On the other hand, the query may also contain randomly perturbed ratings such that the effects of the noise are removed from the noise after the prediction result is obtained. This technique, however, contributes to reduced accuracy.

4 Privacy-preserving Slope One

4.1 Problem statement

The problem can be formally defined in the following fashion:

Definition 1 (Privacy-Preserving weighted Slope One Predictor). *Given a set of m users $u_1, \ldots, u_m$ that may rate any number of n items $i_1, \ldots, i_n$, build the weighted Slope One predictor for each item satisfying the following two constraints:*

- *no submitted rating should be deterministically linked back to any user.*
- *any user should be able to obtain a prediction without leaking his/her private rating information.*

4.2 Additive random noise

In this section, we discuss the effects of various ways of including additive random noise: (1) random noise is added to the weighted Slope One deviation matrix only; (2) random noise is added to the user-item data for the weighted Slope One predictors; (3) in both cases, random noise is added to the query vector;

In our experiments, we also show the situation where random noise is not added to the query vector but the noisy deviations are rounded off to nearest integers to facilitate the use of a public-key cryptosystem in the query, as is the case in our earlier work [10]

Noise added to deviations only In the *pre-computation stage*, random noise (denoted by $\epsilon_{i,a,b}$), obtained from a known probability distribution, added to the deviation of a pair of items (a and b) by a user i generates a noisy deviation, given as:

$$\hat{\delta}_{i,a,b} = r_{i,a} - r_{i,b} + \epsilon_{i,a,b}. \tag{4.1}$$

where $\delta_{i,a,b} = r_{i,a} - r_{i,b}$ is the original deviation of the rating of item a from that of item b both given by user i. Therefore:

$$\hat{\Delta}_{a,b} = \sum_i \delta_{i,a,b} + \sum_i \epsilon_{i,a,b} = \Delta_{a,b} + \sum_i \epsilon_{i,a,b}, \tag{4.2}$$

In the *prediction stage*, adding another similar random noise (denoted as $\nu_{u,a}$) for every item a rated by the user u, the rating for the user u and item x using the *weighted* Slope One on the noisy deviations is predicted as:

$$\hat{r}_{u,x} = \frac{\sum_{a|a\neq x}(\hat{\Delta}_{x,a} + (r_{u,a} + \nu_{u,a})\phi_{x,a})}{\sum_{a|a\neq x}\phi_{x,a}} \tag{4.3}$$

$$\implies \hat{r}_{u,x} = \frac{\sum_{a|a\neq x}\Delta_{x,a} + r_{u,a}\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}} + \frac{\sum_{a|a\neq x}\sum_i \epsilon_{i,a,b}}{\sum_{a|a\neq x}\phi_{x,a}} + \frac{\sum_{a|a\neq x}\nu_{u,a}\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}} \tag{4.4}$$

$$\implies \hat{r}_{u,x} = r_{u,x} + \frac{\sum_{a|a\neq x}\sum_i \epsilon_{i,a,b}}{\sum_{a|a\neq x}\phi_{x,a}} + \frac{\sum_{a|a\neq x}\nu_{u,a}\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}}. \tag{4.5}$$

It is evident from equation 4.5 that the noisy prediction $\hat{r}_{u,x}$ contains small proportions of random noise. For example, if the noise data are drawn from a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma = \textit{ubound}(r))$ (where $\textit{ubound}(r)$ denotes the maximum positive value of the ratings themselves) then the component

$$\frac{\sum_{a|a\neq x}\sum_i \epsilon_{i,a,b}}{\sum_{a|a\neq x}\phi_{x,a}}$$

nearly vanishes and the component

$$\frac{\sum_{a|a\neq x}\nu_{u,a}\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}}$$

is reasonably small for sufficiently large number of data points. This suggests that the prediction accuracy will be better if we added random noise only at the time of pre-computation. Results from our experiments confirm this analysis.

Noise added to ratings only In the *pre-computation stage*, random noise (denoted by $\kappa_{i,a}$ and $\lambda_{i,b}$), obtained from known probability distributions, added to the rating of each item respectively in a pair of items (a and b) by a user i generates a noisy deviation, given as:

$$\hat{\delta}_{i,a,b} = (r_{i,a} + \kappa_{i,a}) - (r_{i,b} + \lambda_{i,a,b}). \tag{4.6}$$

where $\delta_{i,a,b} = r_{i,a} - r_{i,b}$ is the original deviation of the rating of item a from that of item b both given by user i. Therefore:

$$\hat{\Delta}_{a,b} = \sum_i \delta_{i,a,b} + \sum_i(\kappa_{i,a} - \lambda_{i,a}) = \Delta_{a,b} + \sum_i(\kappa_{i,a} - \lambda_{i,a}). \tag{4.7}$$

In the *prediction stage*, adding another similar random noise (denoted as $\iota_{u,a}$) for every item a rated by the user u, the rating for the user u and item x using the *weighted* Slope One on the noisy deviations is predicted as:

$$\hat{r}_{u,x} = \frac{\sum_{a|a\neq x}(\hat{\Delta}_{x,a} + (r_{u,a} + \iota_{u,a})\phi_{x,a})}{\sum_{a|a\neq x}\phi_{x,a}} \tag{4.8}$$

$$\implies \hat{r}_{u,x} = \frac{\sum_{a|a\neq x}\Delta_{x,a} + r_{u,a}\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}} + \frac{\sum_{a|a\neq x}\sum_i(\kappa_{i,a} - \lambda_{i,a})}{\sum_{a|a\neq x}\phi_{x,a}} + \frac{\sum_{a|a\neq x}\iota_{u,a}\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}} \tag{4.9}$$

$$\implies \hat{r}_{u,x} = r_{u,x} + \frac{\sum_{a|a\neq x}\sum_i(\kappa_{i,a} - \lambda_{i,a})}{\sum_{a|a\neq x}\phi_{x,a}} + \frac{\sum_{a|a\neq x}\iota_{u,a}\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}}. \tag{4.10}$$

Similar to adding noise to deviations, we observe from equation 4.10 that the noisy prediction $\hat{r}_{u,x}$ contains small proportions of random noise. Again, if the noise data are drawn from a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma = \mathit{ubound}(r))$ (where $\mathit{ubound}(r)$ denotes the upper bound of the range of rating values) then the component

$$\frac{\sum_{a|a \neq x} \sum_i (\kappa_{i,a} - \lambda_{i,a})}{\sum_{a|a \neq x} \phi_{x,a}}$$

nearly vanishes and the component

$$\frac{\sum_{a|a \neq x} \iota_{u,a} \phi_{x,a}}{\sum_{a|a \neq x} \phi_{x,a}}$$

is reasonably small for sufficiently large number of data points. Once again, this suggests that the prediction accuracy will be better if we added random noise only at the time of pre-computation. In fact, equation 4.10 is better in accuracy than equation 4.5 because the vanishing component

$$\frac{\sum_{a|a \neq x} \sum_i (\kappa_{i,a} - \lambda_{i,a})}{\sum_{a|a \neq x} \phi_{x,a}} < \frac{\sum_{a|a \neq x} \sum_i \epsilon_{i,a,b}}{\sum_{a|a \neq x} \phi_{x,a}}.$$

Do we need noise at the time of the query? Although in both of the above cases we used additive noise at the time of the query, it is possible to not use noise. As we will see in the experimental results, not using noise at the time of query increases accuracy. However, in order to preserve the privacy of the data in the rating query, we will have to use encrypted query with an additively homomorphic cryptosystem as we did in our earlier PPCF proposal in [10]. To facilitate the use of homomorphic encryption, we ought to round off fractional values of deviations before using them in the encrypted response of the query. In our experimental results, we show the effects of such rounding-off on accuracy.

4.3 Multiplicative random noise

Random noise multiplied to the deviation of a pair of items by a user i is given as:

$$\hat{\delta}_{i,a,b} = (r_{i,a} - r_{i,b})\epsilon_{i,a,b} = \delta_{i,a,b}\epsilon_{i,a,b}. \tag{4.11}$$

where $\delta_{i,a,b} = r_{i,a} - r_{i,b}$ is the deviation of the rating of item a from that of item b both given by user i, and $\epsilon_{i,a,b}$ is the noise multiplied to the deviation. Therefore:

$$\hat{\Delta}_{a,b} = \sum_i \delta_{i,a,b}\epsilon_{i,a,b}. \tag{4.12}$$

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

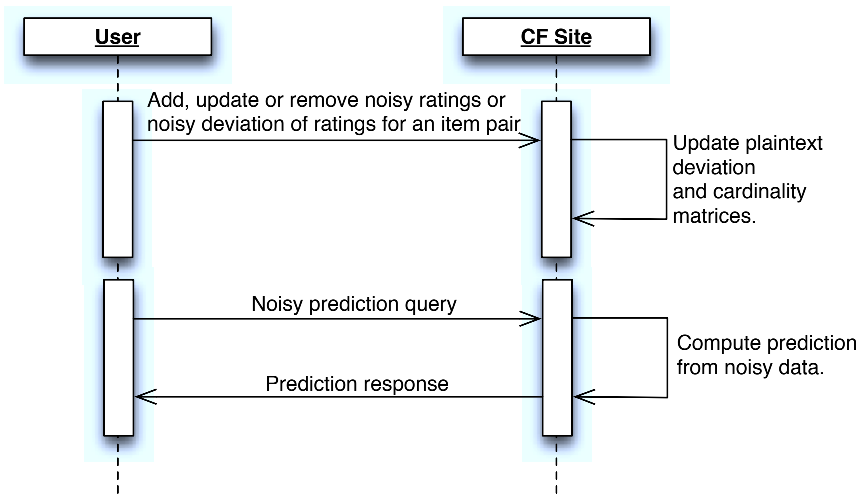
$$\hat{r}_{u,x} = \frac{\sum_{a|a \neq x} (\hat{\Delta}_{x,a} + r_{u,a}\nu_{u,a}\phi_{x,a})}{\sum_{a|a \neq x} \phi_{x,a}} \tag{4.13}$$

We found that result of adding multiplicative noise is that the noise components are significant. Without further reconstruction methods, which are beyond the scope of this paper, the data with random multiplicative noise is unsuitable for use in prediction. Therefore, in the remainder of the paper, we present results only with additive noise.

5 Proposal for privacy-preserving collaborative filtering

5.1 Proposal A: additive noise to ratings or deviations

Having observed the theoretical effects of additive noise, we note that it is more efficient than adding multiplicative noise. One of our PPCF proposals is to have Gaussian noise added to either individual ratings or rating pair deviations as they are added by the users. This helps in masking the actual ratings at the time they are submitted. The CF site can use those submissions to update the deviation and cardinality matrices. At the time of query, once again, the user utilises Gaussian additive noise to hide the actual rating data in the query. The system architecture for PPCF utilising additive noise in this way is presented in the form of UML sequence in figure 5.1.



User
CF Site
Add, update or remove noisy ratings or noisy deviation of ratings for an item pair
Update plaintext deviation and cardinality matrices.
Noisy prediction query
Compute prediction from noisy data.
Prediction response

Fig. 5.1. UML sequence diagram of the proposed PPCF architecture utilising just noise.

5.2 Proposal B: combination of additive noise with encrypted query

In our second proposal, we suggest that while additive random noise can be used at the time of submitting the data to the CF site, the user can also combine that with encrypted queries using public-key encryption. Homomorphically encrypted queries are beyond the scope of this paper but can be found in our proposal on encrypted PPCF in [10]. The system architecture of this PPCF scheme supporting encrypted query is shown in figure 5.2.

One important observation in terms of accuracy in this scheme is that the values of deviations are rounded off to nearest integers because we will need a specialised mechanism to encrypt IEEE floating point numbers with the Paillier cryptosystem. Such a mechanism may greatly reduce the usable length of the key space, which is why we round off the deviations to nearest integers. That will, however, have a small penalty on the accuracy of the prediction as we shall see in the results.

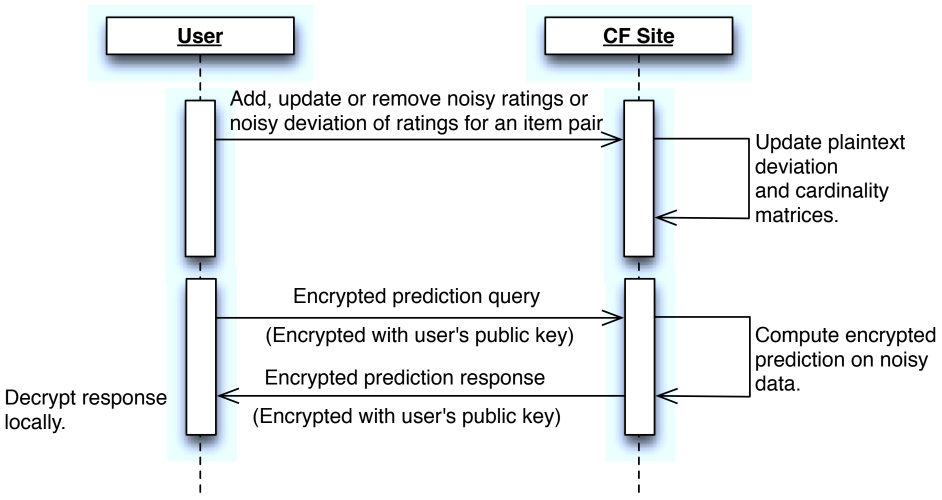

User
CF Site
Add, update or remove noisy ratings or noisy deviation of ratings for an item pair
Update plaintext deviation and cardinality matrices.
Encrypted prediction query
(Encrypted with user's public key)
Compute encrypted prediction on noisy data.
Encrypted prediction response
(Encrypted with user's public key)
Decrypt response locally.

Fig. 5.2. UML sequence diagram of the proposed PPCF architecture utilising noise as well as encrypted queries.

5.3 Level of privacy

A number of methods [11,12,13] exist for quantification of the level of privacy preserved in a given data mining algorithm. Yet, it still an open question, beyond the scope of this paper, as to which one is the best [14]. Therefore, in this paper we qualitatively describe the level of privacy with the additive random noise, as shown in the aforementioned sections. Denoting the random noise variable by ϵ, the original rating variable by r, and the perturbed values by $\hat{r}$, we have $\hat{r} = r + \epsilon$. To an attacker, the distribution of ϵ is known and the distribution of $\hat{r}$ can be observed. The objective is to attempt to reconstruct the distribution of r.

Assuming that ϵ is derived from a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma = k)$ where $k \geq \mathit{ubound}(r)$ ($\mathit{ubound}(r)$ is the upper bound of the range of rating values), we know that the distribution tapers off towards the positive and negative infinities, with majority of the values centered around the mean, i.e. 0 in our case. Now, let us use some concrete numbers. Let us say that $k = 5$ and that for any particular observation, we find $\hat{r} = -40$. Also, assume that the attacker is trying to determine if $r = 1$ for this case. If the attacker had another observation of $\hat{r} = 5$ then, in terms of conditional probability, we can tell $\Pr(\hat{r} = -40 \mid r = 1) < \Pr(\hat{r} = 5 \mid r = 1)$ simply because of the nature of the noise distribution, or even in general $\Pr(\hat{r} = -40 \mid r = 1)$ and in fact $\Pr(\epsilon = -41)$ is low. However, that statement does not imply that $\Pr(r = 1 \mid \hat{r} = -40)$ is equally low, or vice-versa, which is the inverse fallacy [15]. Therefore, the attacker cannot imply that $\Pr(r = 1 \mid \hat{r} = -40) < \Pr(r = 1 \mid \hat{r} = 5)$, because while $\Pr(r = 1 \mid \hat{r} = -40)$ could be low, that could be equal to $\Pr(r = 2, 3, 4, 5 \mid \hat{r} = -40)$ if r was following a uniform distribution. This leaves the attacker with little deterministic ability to infer the value of r. This observation is especially true because of the small bounded range of r and the large unbounded range of ϵ and therefore $\hat{r}$.

Now, from another angle for $\hat{r} = -40$ and $r = 1$, we know $\epsilon = -41$. If $r = 2, 3, 4, 5$ then $\epsilon = -42, -43, -44, -45$. We know that $\Pr(\epsilon = -41)$ is low and so are $\Pr(\epsilon = -41, -42, -43, -44, -45)$ and these probabilities are also very close to each other. Therefore, because of the bounded and small range of r, just looking at any value of $\hat{r}$, it is not possible to determine with the value of r with a high probability.

That proves that the unbounded range of the perturbed values makes it difficult to confidently determine the value of the original rating, without knowing the distribution of the original ratings. This is the scenario where noise is added to the ratings. If the noise is added to the deviation of ratings, the ability to determine the original ratings is decreased further. In addition, since the deviations also have a bounded small range, it is equally hard (as we discussed above) to determine even the value of the deviation from the value of the perturbed deviations when the noise, and therefore the perturbed deviations, have unbounded ranges.

6 Implementation and evaluation

6.1 Implementation

Consideration: reduce I/O operations and storage volumes With the intention to implement our proposals on a real world public cloud computing environment, we took the following considerations in mind. I/O with non-volatile storage is expensive both in terms of time and money on cloud computing platforms. Non-volatile storage on the cloud usually involves in a backend database, which is often replicated (e.g. high-replication datastore in Google App Engine). That implies that one database write operation could entail multiple write operations to database replicas, which in turn increases CPU and bandwidth usage, translating into direct costs to the cloud user. In addition to that, storage space is also charged per unit space per unit time, e.g. gigabytes per month. The larger the storage, the worse the costs. Therefore, it is essential to store as little data as possible.

Keeping this in mind, we have deliberately avoided a particular method of random perturbation: calculating the z-scores of rating data prior to introducing noise, in order to hide the number of items one user might have rated. This essentially transforms a sparse matrix problem into a dense matrix problem, which represents a huge performance hit. Both Amazon RDS and Google GAE/J cost significantly to store data. A dense matrix makes matters worse. For example, the MovieLens 100K dataset, when normalised to z-scores contains over 1.58 million data points compared to the sparse 100,000 in the original rating data. That is an over 1,500% increase in stored data translating directly into that much increase in storage costs in addition to the CPU costs for redundant storage. In fact, we do not store the user-item rating matrix at all but store the sparse deviation and cardinality matrices.

Implementation environment and data structures The results presented in this paper are obtained from a trial, non-cloud, single-machine implementation on Java. We used a 64-bit Mac OS X 10.7.2 and 64-bit Java 1.6.0_29 environment on an Apple Macbook Pro running a 64-bit 2.53GHz Intel Core i5 and 8GB RAM.

The storage of the following two matrices is an important factor in the efficiency of pre-computation as well as prediction: (1) the item-item deviation matrix, and (2) the item-item cardinality matrix. Note that we do not store the user-item ratings matrix at all.

There is significant level of sparseness in the stored data. Although the deviation and cardinality matrices are not as sparse as the user-item ratings matrix, only the upper triangulars of the deviation and cardinality matrices are stored. Using in-memory storage, the sparseness requirement informs us that a 2-dimensional array (e.g. `long[][]`) is an unsuitable storage data structure. While the simple 2-D array provides constant time, i.e. $O(1)$ lookup performance, it is an unjustifiable waste of storage space and makes resizing difficult. The resizable `ArrayList` implementation provides an $O(1)$ lookup performance but $O(n)$ time complexity for the addition operation.

If we access a 2-D matrix by either row-major order or column-major order but not both at the same time then it can be represented by a `Map<K1, Map<K2, V>>`. Java's `TreeMap` implementation provides $O(log\, n)$ lookup and storage performance while `Hashtable` and `HashMap` both provide constant time lookup and storage. Having tested Oracle (Sun) JVM's `HashMap` and `Hashtable` implementations, we found that `HashMap` is faster although there is no theoretical basis supporting this view and may be purely JVM implementation specific. Although mostly similar, one of the differences between `Hashtable` and `HashMap` is that the latter allows `null` values for keys and objects, which is irrelevant in our context. `HashMap` iterator is fail-safe, which means changes made to the map get reflected in its iterator.

Having chosen an efficient data structure, i.e. `HashMap`, we looked at the data types for `K1`, `K2` and `V` because that affects performance too. Both `K1` and `K2` are integers (perhaps `long`) while `V` could contain double precision floating point numbers (`double`) for deviations and `long` for cardinalities. Realistically, the value of the keys is well expressed by Java's primitive 32-bit `int` with a positive range of $[0 \quad (2^{31}-1)]$. This is enough for indexing rows and columns: a $(2^{31}-1)$ by $(2^{31}-1)$ square matrix is extremely large! There is also another advantage of a `HashMap<Integer,V>` because `Integer` provides "a hash code value for this object, equal to the primitive int value represented by this Integer object"[8], which is faster to compute than that for a `Long`, i.e. "`(int)(this.longValue()^(this.longValue()>>>32))`"[9].

In this context, the semantics of an "empty" deviation-cardinality tuple must be understood in order to ensure that the deviation-cardinality matrix allows sparseness by not storing empty tuples. Since the deviation value of zero does not indicate an absence of deviation, "emptiness" is determined by the cardinality value of zero. There is also another point that aids the sparse nature of the matrix – the storage of the upper triangular of the matrix only, discarding the lower triangular and the leading diagonal. While this behaviour is controllable through the matrix access methods, the tuple itself in our implementation enables access to the not-stored lower triangular by inverting the stored value in the upper triangular. Note that the cardinality is not inverted but the deviation (Δ) is, i.e. $\Delta_{i,j} = -\Delta_{j,i}$.

In the context of the cloud Note that the choice of this local in-memory storage data structure somewhat corresponds to storage on the cloud too. For fast access to the deviation-cardinality data on the cloud, we have to use some kind of distributed cache (e.g. memcached) – both Google App Engine for Java and Amazon Web Services provide distributed cache services, which are somewhat similar to the basic `Hashtable<K,V>` structure. Beyond the cache, the distributed databases can also store individual sparse data points using matrix row-column based 2D indexing through database tables, i.e. to store the value at the i^{th} row and the j^{th} column, we simply add a row in the database table that contains the matrix row number (i.e. i), the column number (i.e. j) and any values associated with them. This is the way, we stored data in our earlier PPCF proposal [10].

6.2 Evaluation results

All experiments have been run with the MovieLens 100K dataset. In table 6.1, we present different combinations of our PPCF proposals. We also cite the results of Polat and Du's paper [6], our own work on homomorphic encryption based PPCF [9] and the baseline Slope One for comparison. The different combinations of our PPCF proposals are listed as follows.

[8] See: `http://download.oracle.com/javase/6/docs/api/java/lang/Integer.html`.
[9] See: `http://download.oracle.com/javase/6/docs/api/java/lang/Long.html`.

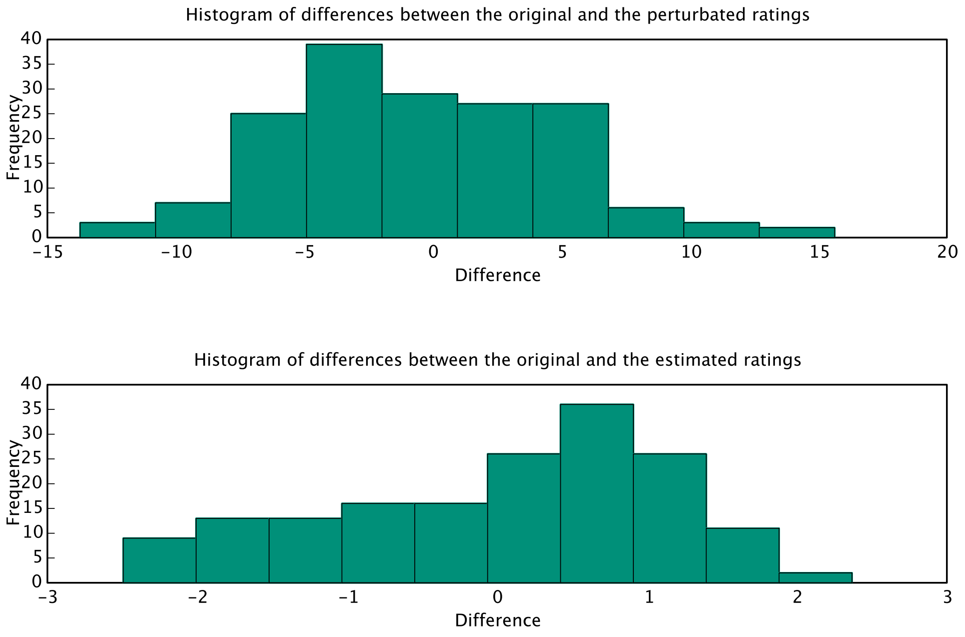


Fig. 6.1. Histograms showing the comparison of the difference between the original and perturbed ratings, and that between the original and the estimated ratings for a random user. In this histogram, we have only compared those values for which the original ratings were present.

A1: This is where Gaussian random noise is added to ratings both at the time of submission and at the time of query.

A2: Here we add Gaussian random noise to deviations of rating pairs at the time of submission and similar noise to ratings at the time of query.

B1: We add random noise to the ratings at the time of submission but we round off deviations (for encryption) at the time of query.

B2: The Gaussian noise is added to deviations of rating pairs but the deviations are rounded off at the time of query.

In all the experiments, the random noise is drawn from a Gaussian distribution $\mathcal{N}(0,5)$.

In figure 6.1, we present a comparison of original ratings, perturbed and estimated ratings for a random user (amongst the 943 users in the MovieLens 100K dataset) using proposal B1. Notice how the majority of the difference between the original and the estimated ratings like lie between 0 and 1, while those between the original and perturbed ratings lie between 0 and -5.

Note that both B1 and B2 are supposed to be used with encrypted queries so the actual prediction time will include homomorphic encryptions and multiplications, hence will be perceptibly slower than what is shown in the results.

Table 6.1. Comparison of results of this work with others using the MovieLens 100K dataset.

	PPCF strategy	MAE	Prediction time	Stored data
Non-PPCF baseline	None	0.7019	0.22ms	Item-item deviation, cardinality matrices.
A1	Perturbation	0.8346	0.23ms	Item-item deviation, cardinality matrices.
A2	Perturbation	0.8307	0.234ms	Item-item deviation, cardinality matrices.
B1	Perturbation	0.7113	0.233ms	Item-item deviation, cardinality matrices.
B2	Perturbation	0.7081	0.231ms	Item-item deviation, cardinality matrices.
Basu et al. [9]	Encryption	0.7057	4500ms	Item-item deviation/cardinality matrices
Polat and Du [6]	Perturbation	0.7104[a]	Unknown	z-scored and randomised user-item rating matrix and its singular value decompositions.

[a] This data provides approximate comparability with our results because of the differences of samples over which the MAE calculations have been made as well as the level of noise added. For example, the noise added in Polat and Du's paper had a standard deviation of 1 whereas we used a standard deviation of 5.

7 Conclusion and future work

Given that the Slope One predictor from Lemire and MacLachlan [2] is robust to additive noise, in this paper, we have proposed some privacy-preserving solutions for the problem of collaborative filtering using random perturbation. Our solution is based on the weighted Slope One predictor and unbounded additive random noise. We have discussed how it is difficult to determine original ratings from the perturbed data while the perturbed data is still good enough for fairly accurate predictions. We have also presented comparative results from a trial implementation and shown design considerations for cloud implementations.

In future, we will explore other randomisation techniques and the applicability of ϵ-differential privacy on the Slope One CF predictors. We will also port our trial implementation to cloud platforms such as Google App Engine for Java and the Amazon Elastic Beanstalk to facilitate practical experimentation at a larger scale.

Acknowledgments

Special thanks to Theo Dimitrakos from British Telecom for his valuable inputs on this paper. 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. Contributions by Theo Dimitrakos relate to research in British

Telecommunications under the IST Framework Programme 7 integrated project OPTIMIS that is partly funded by the European Commission under contract number 257115.

We would also like to thank the anonymous reviewers for their constructive criticism.

References

1. 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
2. Lemire, D., Maclachlan, A.: Slope one predictors for online rating-based collaborative filtering. Society for Industrial Mathematics (2005)
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.C., Yu, P.S.: 2. In: A General Survey of Privacy-Preserving Data Mining Models and Algorithms. Springer (2008) 11–52
8. Basu, A., Kikuchi, H., Vaidya, J.: Privacy-preserving weighted slope one predictor for item-based collaborative filtering. In: Proceedings of the international workshop on Trust and Privacy in Distributed Information Processing (workshop at the IFIPTM 2011), Copenhagen, Denmark. (2011)
9. Basu, A., Vaidya, J., Kikuchi, H.: Efficient privacy-preserving collaborative filtering based on the weighted Slope One predictor. Journal of Internet Services and Information Security **1(4)** (2011)
10. Basu, A., Vaidya, J., Kikuchi, H., Dimitrakos, T.: Privacy-preserving collaborative filtering for the cloud. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (Cloudcom), Athens, Greece. (2011)
11. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: ACM Sigmod Record. Volume 29., ACM (2000) 439–450
12. Agrawal, D., Aggarwal, C.C.: On the design and quantification of privacy preserving data mining algorithms. In: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM (2001) 247–255
13. Evfimievski, A., Srikant, R., Agrawal, R., Gehrke, J.: Privacy preserving mining of association rules. Information Systems **29**(4) (2004) 343–364
14. Evfimievski, A.: Randomization in privacy preserving data mining. ACM SIGKDD Explorations Newsletter **4**(2) (2002) 43–48
15. Villejoubert, G., Mandel, D.: The inverse fallacy: An account of deviations from Bayes's theorem and the additivity principle. Memory & cognition **30**(2) (2002) 171–178