Privacy-preserving targeted advertising

Theja Tulabandhula¹, Shailesh Vaya² and Aritra Dhar³
¹University of Illinois Chicago
²Conduent Labs India
³ETH Zurich

Version: February 12, 2017

Abstract

Recommendation systems form the center piece of a rapidly growing trillion dollar online advertisement industry. Even with optimizations/approximations, collaborative filtering (CF) approaches require real-time computations involving very large vectors. Curating and storing such related profile information vectors on web portals seriously breaches the user’s privacy. While achieving private recommendations in this setup further requires communication of long encrypted vectors, making the whole process inefficient. We present a more efficient recommendation system alternative, in which user profiles are maintained entirely on their device, and appropriate recommendations are fetched from web portals in an efficient privacy preserving manner. We base this approach on association rules.

1 Introduction

Targeted advertising (TA) uses keywords, their frequencies, link structure of web, user interests/demographics, recent/overall buying histories etc. to deliver personalized advertisements. TA is enabled by (unique) cookies (random hash-maps) stored on the user’s devices. When a cookie is retrieved from it, the web-server storing this string can recall the profile of the user associated/stored with it at its end. This mechanism constitutes a serious breach of privacy as it allows the websites to build very elaborate profile of the user at their end ([1, 2]). This leads to the question: can we achieve TA without employing the cookie mechanism? Any such alternate approach would require the user to maintain their profile on their own device, and use it for interactively computing and fetching the appropriate TA from the server. In fact, solving this problem effectively can be considered a highly valuable contribution [3]. We present a solution framework for this very important commercial problem of wide-spread interest. In the presentation, we focus on the crux of the solution: a privacy preserving recommendation system.

We present a novel approach for privacy preserving recommendation system (RS), which shifts most of the computational work to a pre-processing stage. In the query processing phase, most of the computation is done by the server, while the client merely computes and exchanges encryptions of a few messages. Our system is based on selection and application of association rules (AR), to produce an ordered list of recommended items. ARs [4] capture the relation that if a user has already bought a set of items $p$ (called the antecedent), then she is very likely to also buy another set of items $q$ (called the consequent). ARs are mined from large databases of user purchase data, and often filtered to retain only the most meaningful insights [5]. While CF methods [6] extract and use pairwise marginal statistics from the
joint probability distribution of items, ARs take into account more complex information like functions of the joint probability distributions of much larger set of items considered together. Most computation in CF based system is done online when the recommendation is computed. However, ARs are generated at pre-processing stage, and query-processing merely requires selection and application of the right AR interactively. This process has low communication complexity. Lastly, CF encounters more bottlenecks when privacy preserving properties are desired: (a) Explicit and consistent feedback from users are hard to obtain, store, organize and compile, given that they are to be attached to right user ID. (b) User’s spatio-temporal profile changes continuously and inaccurate predictions result from CF.

We devise meaningful criteria for selection of the most relevant ARs to an ordered list of items. We present fast exact and approximate algorithms satisfying these criteria, and their privacy preserving versions in which the client does not reveal its transaction to server, and the server does not reveal its database of ARs to client. We present experimental results to demonstrate the practicality of our solutions in e-commerce, and other domains.

**Formalization.** There is a client $C$, which has a set of items, referred to as a transaction set $T \subset 2^I$, with $I$ being the universal item set. A server $S$ processes a large transaction database and stores a list of $D > 0$ association rules, in the form of $\{p_i \rightarrow q_i\}_{i=1}^D$. Given $T$, the server $S$ computes and sends recommended items that are based on antecedents of matching association rules, where matching is defined suitably: If there are multiple ARs that are applicable to the transaction, then an ordered list of recommended items is prepared by collating the items recommended by each of the multiple ARs, taking into consideration the weights attached to the ARs (for instance, these can be lift, conviction, Piatetsky-Shapiro etc.), or the items in the input transaction (which may be assigned a weight as according to some monotonously decreasing function of time lapsed since this item was purchased).

**Overview of contributions.** Given an input transaction $T$, defined as an ordered list of items, held by $C$, and a database of ARs ($D$) held by $S$, $C$ and $S$ are to privately and interactively compute the most relevant item, or ordered list of items, to be recommended to $C$.

1. **Criteria:** We formulate several criteria for selecting the appropriate set of association rules. A rule is applicable if its antecedent is contained in the transaction. These criteria are differentiated using parameters such as threshold weight $w$, which is used to eliminate all rules below the given threshold weight, antecedent length threshold $t$, which is used to eliminate all rules above the given length threshold, and parameter $k$, which is used to select the top-$k$ association rules under a specified ordering. We define a class of new search criteria called Generalized Subset Containment Search (GSCS). We show how the GSCS criteria are a strict super-set of the Maximum Inner Product Search (MIPS) criterion [7].

2. **Algorithms:** We present efficient exact and approximate algorithms for computing recommended items based on the criteria above. (a) Exact implementations build on a novel two-level hashing based data structure that stores the rules in a manner so that their antecedents can be appropriately matched, and corresponding consequent(s) can be efficiently fetched. The implementations are parallelizable, and can exploit multi-threading machines. (b) We also design a novel fast randomized approximation algorithm for fetching applicable ARs building on Locality Sensitive Hashing (LSH) [8], and hashing based algorithms for MIPS [9].

3. **Privacy preserving 2-party protocols:** We develop efficient privacy preserving protocols corresponding to the above exact and approximate algorithms. The protocol for the approximate algorithm can be easily extended to embed many other large scale data processing
tasks that rely on hashing (for instance, the record linkage, data cleaning and duplicity detection problems \cite{10} to name a few). Finally, we extensively evaluate the impact that adding privacy has in terms of latency in recommending items. We emphasize that these latencies are manageable for reasonable sized databases and practical for the targeted advertising setting.

**Related Work.** The GSCS problem introduced in Section 3 is similar to other popular search problems on sets including the Jaccard Similarity (JS) problem and vector space problems including the Nearest-Neighbor (NN) and the Maximum Inner Product Search (MIPS) problems. For these related problems, solutions based on LSH are already available. For instance, one can find a set maximizing Jaccard Similarity with a query set using a technique called Minhash. The NN problem can be addressed using L2LSH \cite{8} and variants. The MIPS problem can be solved approximately using methods such as L2-ALSH(SL \cite{7} and Simple-LSH \cite{9}. Finally, note that GSCS is different from the similar sounding Maximum Containment Search problem defined in \cite{10}. The latter is equivalent to the MIPS problem while the former is not.

Privacy preserving recommendation systems have been well studied in the past. For instance, privacy based solutions for different types of collaborative filtering systems have been proposed in \cite{11, 12, 13}. Roughly in that setting, the problem reduces to computing the dot product of a matrix with a vector of real numbers, where the (recommendation) matrix is possessed by the server and the client possesses the vector, and both client and server are interested in preserving the privacy of their data. Since embedding such schemes into a privacy protocol based on cryptography is difficult, many solutions resort to data modification and adding noise. For instance, in \cite{14}, the authors propose a perturbation based method for preserving privacy in data mining problems. This approach is only applicable when one is interested in aggregate statistics and does not work when more fine-grained privacy is needed. In \cite{15}, propose a decentralized distributed storage scheme along with data perturbation to achieve certain notions of privacy in the collaborative filtering setting. For the same setting, a method based on perturbations is also proposed in \cite{16} and \cite{17}. In the paper \cite{18}, the authors proposed a theoretical approach for a system called Alambic which splits customer data between the merchant and a semi-trusted third party. The security assumption is that these parties do not collude to reveal customers’ data. A major difference between our work and all these solutions is that we base our privacy solutions based on cryptography and use the framework of association rules that are heavily already used in practice. In particular, we propose some of the first practical distributed privacy preserving protocols for recommendation systems based on selection and application association rules. Note that privacy has also been well studied in the context of generation of association rules \cite{19, 20}, but not much for the problem of their selection and application.

### 2 Recommendation Criteria

We have a set of $D$ association rules $\{p_i \rightarrow q_i\}_{i=1}^D$ with $p_i, q_i \subseteq 2^I$, and additional attributes (for instance, interestingness measure $w_i$). Given a query transaction $T \subseteq 2^I$ of the client, we perform two steps: a Fetch operation followed by a Collate operation. We propose multiple criteria for deciding which association rules should be selected. We also define a search problem called the Generalized Subset Containment Search (GSCS) and relate it to one of the criteria. Our algorithm in Section 3 solves the GCS problem in a computationally efficient manner.

**Fetch step: Selection of rules.** We define that an association rule $i$ is *applicable* to a trans-
Uncapacitated setting: We simply return the union of consequent(s) the client may have a constraint, e.g., \( f \) such as threshold weight \( w \in \mathbb{Z}_+ \), which is used to eliminate all applicable association rules below the threshold weight, \( t \), which is used to eliminate rules with antecedent lengths greater than \( t \in \mathbb{Z}_+ \), and \( k \in \mathbb{Z}_+ \), which is used to select the top \( k \) association rules according to a predefined notion of ordering specified using a function \( f : \{1, \ldots, D\} \rightarrow \mathbb{Z} \).

The TOP-Assoc \((k, w, t, f)\) criterion outputs a set of applicable rules base on three parameters and an ordering function \( f \). Parameter \( w \) filters out rules with weights \( \leq w \). Parameter \( t \) retains rules with antecedents of length \( \leq t \). Parameter \( k \in \mathbb{Z}_+ \) controls the number of applicable rules that are finally output. We can write the following optimization problem representing this criterion as follows: \( \max_{x \in \{0,1\}^D} \sum_i x_i : f(i) \) such that \( \sum_i x_i \leq k \), and \( x_i \leq \min\{1[|p_i| \leq t], 1[w_i \geq w], 1[p_i \subseteq T]\} \). MAX-Assoc \((f)\) criterion is a special case of TOP-Assoc \((k, w, t, f)\) criterion where \( k = 1 \), \( w = 0 \) and \( t = |\mathcal{I}| \). ALL-Assoc \((w, t)\) criterion is another special case where \( k = D \).

Under the ANY-Assoc \((k, w, t)\) criterion, the output contains at most \( k \) applicable association rules with weights \( \geq w \) and rule antecedent lengths \( \leq t \). The corresponding formulation is \( \max_{x \in \{0,1\}^D} \sum_i x_i : f(i) \) such that \( \sum_i x_i \leq k \), and \( x_i \leq \min\{1[|p_i| \leq t], 1[w_i \geq w], 1[p_i \subseteq T]\} \).

Ordering functions. The function \( f \) determines which of the applicable rules are the top \( k \) rules. We can define \( f \) such that rules can be ordered according to: (a) their weights \( w_i \) (e.g., \( f(i) = w_i \)), or (b) antecedent lengths \( |p_i| \) (e.g., \( f(i) = |p_i| \)) or, (c) a combination of both (e.g., \( f(i) = g_1(w_i) + g_2(\max) \cdot g_2(|p_i|) \)), where \( g_1 \) and \( g_2 \) are strictly monotonic integer-valued functions and \( \max = \max_{i=1, \ldots, D} w_i \). For a pair of rules with antecedents \( p_1 \) and \( p_2 \) and weights \( w_1 \) and \( w_2 \), this latter function has the following properties: (i) If \( |p_1| < |p_2| \), then \( f(1) \leq f(2) \), and (ii) If \( |p_1| = |p_2| \) and \( w_1 \leq w_2 \), then \( f(1) \leq f(2) \). Another example of a combination ordering function is \( f(i) = g_2(|p_i|) + g_2(|\mathcal{I}|) \cdot g_1(w_i) \), which prefers weights and then lengths in case of ties.

Collate step: Item recommendations. Once we have generated a set \( \mathcal{L} \) of applicable association rules according to one of the criteria described above (assumed non-empty, otherwise we return a predefined list such as the list of globally most frequent items), we can compile a list of item recommendations in the following two ways:

Uncapacitated setting: We simply return the union of consequent(s) \( q_i \) of the association rules in \( \mathcal{L} \); however, this list can be potentially large.

Capacitated setting: The client may have a constraint \( k' << |\mathcal{I}| \) on the number of items it can recommend to the user. In this case, we derive associated weights \( \bar{w}_j \) for each item \( j \in \bigcup_{i \in \mathcal{L}} q_i \) by adding up the weights \( w_i \) of the rules where item \( j \) is in the consequent \( q_j \). The list of recommended items are then sorted according to these accumulated weights. If the bound \( k' < |\bigcup_{i=1}^{|\mathcal{L}|} q_i| \), we return the top \( k' \) items from this sorted list, else we return all the items.

Generalized Subset Containment Search. The MAX-Assoc \((f)\) criterion with an ordering function preferring longer rules leads to two new search problems: (a) the Largest Subset Containment Search (LSCS) problem, and (b) its generalization, the Generalized Subset Containment Search (GSCS) problem.

Largest Subset Containment Search problem The problem \( \max_{i \in D} \sum_{j=1}^{|\mathcal{I}|} \mathcal{T}^j \cdot p_i^j \) subject to \( p_i^j \leq \mathcal{T}^j \quad 1 \leq j \leq |\mathcal{I}| \) attempts to find a set (i.e., the characteristic vector) whose inner product with the \( \mathcal{T} \) vector is the highest among all sets that are subsets of \( \mathcal{T} \). It is related to the MAX-Assoc \((f)\) criterion as shown below.

Lemma 1. When \( f(i) = 1 \) for all \( 1 \leq i \leq D \), the MAX-Assoc \((f)\) criterion is equivalent to the LSCS problem. Further, if \( p_i \subseteq \mathcal{T} \), then \( \|p_i\|_1 \) (the \( \ell_1 \)-norm of \( p_i \) vector) is equal to \( \sum_{j=1}^{|\mathcal{I}|} p_i^j \mathcal{T}^j \) (here, \( p_i^j \) is the \( j \)-th coordinate of \( p_i \)).
The lemma lets us design an sub-linear time (\( o(D) \)) approximate algorithm for fetching (see Section 3). In particular, we build on unconstrained inner-product search techniques [9] that solve the Maximum Inner Product Search (MIPS) problem: given a collection of “database” vectors \( r_i \in \mathbb{R}^d, 1 \leq i \leq D \) and a query vector \( s \in \mathbb{R}^d \) (\( d \) is the dimension), find a data vector maximizing the inner product with the query: \( r^* \in \arg \max_{1 \leq i \leq D} \sum_{j=1}^d r^*_i s^j \).

Note that it is not straightforward to apply MIPS to solve LSCS. To see this, consider the following MIPS instance constructed to mimic an LSCS instance. Let \( d = |I| \). Let \( r_i \) be equal to normalized antecedent vectors: \( r_i = \frac{1}{\|p_i\|} p_i \), where \( p_i \in \{0,1\}^{|I|}, 1 \leq i \leq D \), and let \( s \) be equal to \( T \in \{0,1\}^{|I|} \). This normalization ensures that smaller length antecedents are preferred in the MIPS instance in order to mimic the subset containment or ‘applicable’ property. Let \( p_{LSCS} \) and \( p_{MIPS} \) be the optimal solutions of the LSCS and MIPS instances constructed above.

In other words, part (1) implies that if the LSCS instance is feasible, then there exists an optimal solution \( p_{MIPS} \) for the constructed MIPS instance that has \( p_{MIPS}^j = 0 \) for all coordinates where \( T^j = 0 \). Part (2) implies that the optimal solutions of the MIPS instance are potentially feasible for the LSCS instance if they satisfy a condition. However, there is no guarantee that they will be optimal for the LSCS problem. In the worst case, there could be as many as \( O(2^{|I|}) \) optimal solutions for the MIPS instance but only a unique solution for the LSCS instance.

**Generalized Subset Containment Search:** LSCS can be generalized to get the GSCS problem: \( \max_{1 \leq j \leq |I|} f(i) \cdot \sum_{j=1}^{|I|} T^j \cdot p^j \) subject to \( p^j \leq T^j \) \( 1 \leq j \leq |I| \), where \( f(i) \) is the ordering function. When \( f(i) = 1 \), this is LSCS. In other words, for LSCS the ordering determining the top is not dependent on attributes such as the weight \( w_i \); it is only dependent on the antecedent length (through the inner product term). On the other hand, GSCS can account for arbitrary ordering functions. Since GSCS problem is more general, it is clear that GSCS and MIPS are also different.

### 3 An approximate algorithm

We describe a new approximate randomized algorithm to fetch a highly applicable association rule, corresponding to the MAX-ASSOC criterion under an ordering preference that **prefers applicable rules with large antecedents.** In particular, the objective is to return the rule with the largest antecedent set contained within the query set \( \mathcal{T} \) if it is unique, else to return the rule with the highest weight among rules with the largest contained antecedents. This preference ordering leads to a GSCS instance and our algorithm solves this instance by constructing an approximating MIPS instance and solving it using refinements of an LSH based MIPS solver proposed in [9].

The scheme has two parts: (a) APPROX-GSCS-PREP (Algorithm 1), and (b) APPROX-GSCS-QUERY (Algorithm 2). In APPROX-GSCS-PREP, the algorithm prepares a data structure based on all rules that can be efficiently searched at query time. In APPROX-GSCS-QUERY, the algorithm is given a query and returns the rule with the highest weight among rules with the largest contained antecedents. If \( \mathcal{T} \) is unique, then the rule is returned. Otherwise, the algorithm returns the rule with the highest weight among those with \( \mathcal{T} \) as antecedents.
if for any query \( q \): Recall the following definition, which defines properties of hashing:

\[
\text{The approximate MIPS instance we construct is: } \max_{i \in D} f(i) \cdot \sum_{j=1}^{\vert I \vert} p_i^j T^j \text{ such that } p_i^j \leq T^j \text{ for all } 1 \leq i \leq D, 1 \leq j \leq \vert I \vert.
\]

The ordering function can be any function that only comes into effect for breaking ties between vectors \( a \) and \( b \).

**The Approximate MIPS:** The approximate MIPS instance we construct is:

\[
\max_{i \in D} f(i) \cdot \sum_{j=1}^{\vert I \vert} p_i^j T^j
\]

where we have replaced the hard constraints related to subset containment with a proxy that prefers antecedents with larger length. The objective of the MIPS instance can be viewed as an inner product between two real vectors, where the first vector is \( f(i) \cdot \sum_{j=1}^{\vert I \vert} p_i^j \in \mathbb{R}^{\vert I \vert} \), then we can ensure that the scaled vector satisfies \( \|p_i^j\|_2 \leq 1 \). Thus, the constant scaling helps us achieve two things: (a) using \( p_i^j \) in the MIPS instance has the same effect as using \( \frac{f(i)}{\|p_i\|_1} p_i \), and (b) because its \( \ell_2 \) norm is smaller than 1, we can apply the technique proposed in \[9\] to build our rule data structure as shown below.

**The data structure:** Recall the following definition, which defines properties of hashing functions that are used in many retrieval solutions \[8\].

**Definition 1.** For a domain \( D \) of points, a family \( \mathcal{H} = \{ h : D \rightarrow \mathcal{U} \} \) is called locality-sensitive, if for any query \( q \), the function \( \mathcal{P}_h[q, v] = \text{sim}(q, v) = t \) is strictly increasing in \( t \). Here \( \text{sim}(a, b) \) measures the similarity between two points \( a \) and \( b \).

If we could hash our data points (say \( v_i \in D : 1 \leq i \leq D \)), then at query-time we can output the points with which the hash of query \( q \) collides. Our scheme also works on this principle. In general, the similarity measure determines the hashing functions. The subroutine that we use, namely \text{SIMPLE-LSH-PREP} \[9\], relies on the inner product similarity measure and comes with corresponding guarantees on the retrieval quality. To construct a data structure for fast retrieval, it uses hash functions parametrized by spherical Gaussian vectors \( a \sim \mathcal{N}(0, I) \) such that \( h_a(x) = \text{sign}(a^T x) \) (\text{sign()} is a scalar function that outputs +1 if its argument is positive and 0 otherwise).

Given the scaled \( p_i^j \) vectors, \text{SIMPLE-LSH-PREP} constructs a data structure \( DS \) as follows. It defines a mapping \( P \) for vector \( x \in \{ x \in \mathbb{R}^{\vert I \vert} : \|x\|_2 \leq 1 \} \) as:

\[
P(x) = \left[ x; \sqrt{1 - \|x\|_2^2} \right] \in \mathbb{R}^{\vert I \vert + 1}.
\]

Then, for any \( p_i^j \), due to our scaling, we have \( \|P(p_i^j)\|_2 = 1 \). Let \( T' = \frac{\|\|P(p_i^j)\|_2\|}{\|\|P(p_i^j)\|_2\|} T \) be the scaled version of the transaction \( T \). Then, for any scaled vector \( p_i^j \) and \( T' \) we have the following property:

\[
P(p_i^j)^T P(T') = \frac{f(i)}{\|p_i\|_1 \max_{j \geq 1} \sum_{j=1}^{\vert I \vert} p_i^j T^j} \sum_{j=1}^{\vert I \vert} p_i^j T^j,
\]

where \( (\cdot)^T \) represents the transpose operation. This implies that the inner product in the space defined by the mapping \( P \) is proportional to our MIPS instance objective. Further, in this new space, it has been shown \[9\] that using the hash functions \( \{h_a\} \) defined earlier to perform fast Euclidean nearest neighbor search achieves doing an inner product search in the space defined by the domain of \( P \).

\text{SIMPLE-LSH-PREP} uses two additional parameters, namely, a concatenation parameter \( K \), and a repetition parameter \( L \), to amplify the gap between the collision probabilities of similar
points and dissimilar points. SIMPLE-LSH-PREP picks a sequence of $K \cdot L$ hash functions from $\{h_a\}$ and gets a $K \cdot L$ dimensional signature for each vector $P(p'_i) \in \mathbb{R}^{I+1}$, $1 \leq i \leq D$. These signatures and the chosen hash functions are output as $DS$.

**Algorithm 1: Approx-GSCS-Prep($\{p_i, w_i\}, f, L, K$)**

**Input:** $D$ rules with antecedents $p_i \in \{0, 1\}^I$ and weights $w_i \in \mathbb{Z}_+$, ordering function $f$, hashing concentration parameter $K$ and hashing repetition parameter $L$.

**Output:** Data Structure $DS$ containing rule representations and hashing constants

1. Begin
2. Define $f_{\text{max}} = \max_{i \in D} f(i)$.
3. Do parallel{
   4. forall $i = 1, \ldots, D$ do
      5. $p'_i \leftarrow f(i) \| p_i \|_{f_{\text{max}}} p_i$
   }End parallel
7. $DS \leftarrow$ Simple-LSH-Prep($\{p'_i\}_{i=1}^D, K, L$)
8. Return $DS$

**Query.** Given a transaction $T$, APPROX-GSCS-QUERY queries SIMPLE-LSH-QUERY (pseudo-code in [9]) to obtain candidates that are potentially the top rules. SIMPLE-LSH-QUERY solves the similarity problem in sub-linear time by filtering out most similar neighbors of transaction vector $T$ in $\{p'_i : 1 \leq i \leq D\}$. It does this by constructing the vector $[T; 0] \in \mathbb{R}^{I+1}$ and getting its $K \cdot L$ dimensional signature with the same hash functions as used before for $p'_i$ vectors. Then it collects rules that share the same signature as the transformed transaction vector. In particular, it ensures that the signatures agree in at least one $K$-length chunk out of the $L$ chunks. Appropriate choices of $K$ and $L$ (which do not depend on the number of rules $D$) allows for retrieval of the top candidates with high approximation quality.

APPROX-GSCS-QUERY prunes this candidate list to ensure that there is no rule whose antecedent is not a subset of the query (this is a linear search with worst case time complexity $2^{|T|}$). After pruning, there are two cases: (a) no rule is left: in this case, it returns some predefined baseline rule, or (b) some rules are left: in this case, it orders these according to their antecedent length and $f(i)$ (in case of ties) and outputs the top most one.

**Algorithm 2: Approx-GSCS-Query($DS, T$)**

**Input:** Data structure $DS$ from APPROX-GSCS-Prep, query $T \in \{0, 1\}^I$.

**Output:** Top most rule

1. Begin
2. Set $S \leftarrow \phi$
3. $S \leftarrow$ SIMPLE-LSH-QUERY($T, DS$)
4. Set $S' \leftarrow \phi$
5. forall $i \in \{i : p'_i \in S\}$ do
   6. if $p_i \subseteq T$ then $S'.add(i)$
7. if $S' = \phi$ then
   8. return a pre-defined default rule
10. else
   11. Sort the rules in $S'$ according to $f$ in decreasing order
   12. return the rule at the top of the list

**Performance optimization by query scaling:** Scaling $T$ to $T' = \frac{1}{\|T\|_2} T$ as mentioned in preprocessing is not necessary. In particular, the following Lemma shows that our proposed change in processing the query is equivalent to the processing where a normalization is carried out [9]. The advantage of this change is that we do not have to work with a real-valued vector at
query time, leading to an efficient OT step in the privacy preserving counterpart (see Section 5). Recall function \( P(x) = [x; \sqrt{1 - \|x\|^2_2}] \) for \( x \in \{x \in \mathbb{R}^d : \|x\|_2 \leq 1\} \).

**Lemma 3.** Given a Gaussian vector \( a \in \mathbb{R}^{d+1} \) and a transaction vector \( T \in \{0, 1\}^d \), \( \text{sign}(a^T \mathcal{P}(\frac{T}{\|T\|_2})) = \text{sign}(a^T [T; 0]) \).

### 4 The exact algorithms

For exact retrieval of applicable association rules according to any of the four criteria (Section 2), we perform a linear scan over all rules, filter according to the appropriate thresholds and sort them according to an ordering function. Our main contribution here is a data structure that has two attractive properties: (a) It can efficiently store the rules for fetching quickly, and (b) It is easy to privatize for query retrieval (Section 5). By easy to privatize, it meant that the data structure helps in allowing a client to quickly verify whether a string \( Str \) exists in the database \( D \), while revealing little information about \( Str \) to server. The query processing algorithm retrieves any data associated (in our case, consequent \( q \) of AR \( p \rightarrow q \)) with \( Str \), without client revealing it to server, and with server revealing minimum information about its database \( D \) of ARs. Lastly, and most importantly the data structure is efficient in terms of the overall communication complexity, round complexity as well as computations done by the client and the server.

**Pre-processing.** This data structure \( \mathcal{H} \) is common to exact implementations of all four criteria. We describe the structure in a generic way for retrieval of strings. Adapting it to rules (and their attributes) is straightforward. Let \( \Sigma \) be a set of symbols and let \( \Sigma^* \) be the set of all strings over \( \Sigma \). We are given a database \( D \) of strings, with each string of maximum length \( M \). Data structure \( \mathcal{H} \) (generated by Exact-Fetch-Prep, given in Algorithm § 3) is an adaptation of [21], but unlike [21] it is symmetric and has two levels. By symmetric we mean that a fixed hash function \( (h_r) \) will be chosen for hashing all elements at the first level, and another hash function \( (h_s) \) is chosen for hashing all elements at the second level. These hash functions map elements from \( D \) (also representing the size of database) to a range of size at most \( L = 16 \cdot D \) and are chosen hash randomly from a 2-Universal hash function family [22] \( \mathcal{H}_2 = h : [U] \rightarrow [L] \), where \( U \) is a large positive integer.

Additionally, we first choose a couple of large integers \( r \) and \( l \), a string \( r' \) of length \( l \) and the MD5 hash function [23] \( (C_r \), with domain \( \Sigma^{M+l} \) and range \( 2^r \)) to do some pre-processing on the database strings. Once \( \mathcal{H} \) is created on the server, it publicly declares the hash functions \( h_r \) and \( h_s \) as well as the constant string \( r' \) it generated.

We show that Exact-Fetch-Prep identifies the two hash functions \( h_r, h_s \) in fast expected polynomial time and uses \( O(D \cdot D + D) = O(D^2) \) storage (because we chose \( L = 16 \cdot D \)). In particular, building on the analysis in [21], the probability for random hash functions \( h_r \) and \( h_s \) to be successful in the first and second stages of Exact-Fetch-Prep can be bounded as follows.

**Lemma 4.** (1) For \( h_r \in \mathcal{H}_2 \), \( \text{Pr}[\sum_{i=1}^{16D} b_i^2 \leq 4D] \geq \frac{1}{2} \). (2) Function \( h_s \in \mathcal{H}_2 \) succeeds with probability \( \geq 3/4 \).

**Lemma 5.** The data structure \( \mathcal{H} \), can be constructed by Exact-Fetch-Prep in randomized polynomial time.

**Query.** To query a string \( str \), a client can compute the following quantities: \( x = r' \circ \text{str} \), and index \( i = L \cdot h_r(x) + h_s(x) \). It then fetches the indexed element \( \mathcal{H}[i] \) including one of its
Algorithm 3: Exact-Fetch-Prep(D): Creating two-level data structure H

| Line | Action |
|------|--------|
| 1    | **Input:** Database D  |
| 2    | **Output:** Hash table with two level hashes H |
| 3    | begin |
| 4    | Choose: (a) large positive integers r and l, (b) arbitrary string r' of length l, and (c) collision resistant cryptographic hash function \( C_r : \Sigma^{M+l} \rightarrow 2^r \).  |
| 5    | forall \( x \in D \) do  |
| 6    | \( h_r \sim \text{Uniform}(H_2) \)  |
| 7    | forall \( i = 1, \ldots, L \) do  |
| 8    | \( B_i = \{ x \in D : h_r(x) = i \} \)  |
| 9    | \( b(i) = |B_i| \)  |
| 10   | while \( \sum_{i=1}^{L} b_i^2 \leq 4|D| \) do  |
| 11   | \( h_s \sim \text{Uniform}(H_2) \)  |
| 12   | forall \( 1 \leq i \leq L \) and forall \( x, y \in B_i, h_s(x) \neq h_s(y) \) do  |
| 13   | Initialize array \( H \) of size \( L^2 \).  |
| 14   | forall \( x \in D \) do  |
| 15   | \( H[\{ L \cdot h_r(x) + h_s(x) \}] = x \)  |
| 16   | return \( (H, h_s, h_r, r') \)  |

attributes \( H[i], C_r(x') \). This attribute can be used to verify if \( str \) was indeed present in the database. Exact-Fetch-Query (Algorithm §4) implements this query process.

Algorithm 4: Exact-Fetch-Query(str): Query data structure H

| Line | Action |
|------|--------|
| 1    | **Input:** Query string \( str \), and data structure \( H \) from Algorithm §3  |
| 2    | **Output:** Value in \( H \) corresponding to query |
| 3    | begin |
| 4    | Client computes \( x = r' \circ str, h_r(x) \) and \( h_s(x) \).  |
| 5    | Client computes \( i = L \cdot h_s(x) + h_r(x) \).  |
| 6    | Client asks server to return entry at index \( i \) in \( H \).  |
| 7    | Server returns \( H[i] \).  |

If a client sends a query for an element \( x \) to a server, then the properties of the data structure \( H \) allow for limited privacy at query time as claimed below.

**Lemma 6.** (1) The client does not learn about the presence or absence of any other element in \( D \), other then whether \( x \in D \). (2) The server learns only the hash value \( h_r(r' \circ x) \) and \( h_s(r' \circ x) \) of the element \( x \), and nothing more about the actual element \( x \).

5 Privacy preserving protocols

Consider the following two party task: a client \( C \) has an index \( i \), and a server \( S \) has a database \( D \) represented as a vector \( \vec{v}[1 : n] \) of \( n \) elements. The client should fetch the \( i^{th} \) element \( \vec{v}[i] \) such that: (a) the client learns nothing more than the element it fetched from the server, and (b) the server learns nothing about client’s query. Specifically, this leads to the following definition for oblivious transfer (OT):

**Definition 2.** An oblivious transfer(OT) protocol is one in which \( C \) retrieves the \( i^{th} \) element from \( S \) holding \([1, \ldots, n]\) elements iff the following conditions hold:

(1) The ensembles \( \text{Views}_S(S(\vec{v}),C(i)) \), \( \text{Views}_S(S(\vec{v}),C(j)) \) are computationally indistinguishable for all \((i, j)\), where the random variable \( \text{View}_S \) refers to the transcript of the server created by execution of the protocol.
There is a (probabilistic polynomial time) simulator \( \text{Sim} \), such that for any query element \( c \), the ensembles \( \text{Sim}(c, \overline{T}[c]) \) and \( \text{View}_C(S(\overline{T}), C(c)) \) are computationally indistinguishable.

We use the notation \( \text{OT}[\mathcal{C} : i, \mathcal{S} : [1, \ldots, n]] \) to represent the above Protocol Definition §2. We deploy a fast and parallel implementation of OT described in [24]. This scheme is based on length preserving additive homomorphic encryption, described next. Homomorphic encryption with public key \( \text{pk} \), of message \( m \), is denoted as \( c = E_{\text{pk}}(m) \). Decryption with private key \( \text{sk} \) is denoted as \( m = D_{\text{sk}}(c) \). Any operation over the cypher text, will also reflect the decrypted plain text. For instance, let \( c_1 \) and \( c_2 \) be two cipher texts such that \( c_1 = E_{\text{pk}}(m_1) \) and \( c_2 = E_{\text{pk}}(m_2) \). Let + represent a binary operation. Then, \( c_1 + c_2 = E_{\text{pk}}(m_1 + m_2) \). Further-more, the scheme is length preserving so that an \( l \)-bit input is mapped to an input of size \( l + c \), where \( c \) is a constant.

**Private protocol for answering if \( \text{str} \in D \):** Recall the data structure \( \mathcal{H} \) output by \text{Exact-Fetch-Prep} in which \( \mathcal{C} \) fetches an element from database stored with \( \mathcal{S} \). We now present a stronger privacy preserving protocol for the same, by employing OT Definition 2. Thus, by a single execution of OT between \( \mathcal{C} \) and \( \mathcal{S} \), \( \mathcal{C} \) can privately fetch the \( \mathcal{C}_r \)-hash of string \( \text{str} \circ \text{r}' \), stored in record \( \mathcal{H}[D \cdot h_r(\text{str} \circ \text{r}')] + h_s(\text{str} \circ \text{r}')] \cdot \mathcal{C}_s() \). Thus, the choice of using a \( \mathcal{C}_r \)-hash leads to the following guarantee on the OT based protocol.

**Lemma 7.** There exists a two party protocol \text{Private-Exact-Fetch-Query} such that: (1) \( \mathcal{C} \) learns whether \( \text{str} \in D \) with very high probability given the description of associated hash functions \( (h_r, h_s) \), and (2) the computationally bounded \( \mathcal{S} \) learns nothing.

**Remark 1.** A plausible way for \( \mathcal{S} \) to even hide the descriptions of the hash functions \( h_r, h_s \) is for \( \mathcal{C} \) to encrypt \( \text{str}_r = \text{str} \mod r \), \( \text{str}_s = \text{str} \mod s \), using appropriately chosen additively chosen homomorphic encryption, and sending it to \( \mathcal{S} \). \( \mathcal{S} \) computes the encryptions of \( a_r \cdot \text{str}_r + b_r \cdot p_r \cdot r \), and \( a_s \cdot \text{str}_s + b_s \cdot p_s \cdot s \) from the received encrypted values (for randomly chosen \( p_r, p_s \) within appropriate range), and sends these values to \( \mathcal{C} \). \( \mathcal{C} \) extracts the respective values to obtain the respective indexes, and then fetches the final value by executing the OT with \( \mathcal{S} \).

**Private protocol for collating selected rules:** In collating, a client computes an ordered list of recommended items from the set of consequent-s of all applicable association rules.

Firstly, note that only a few items from the client’s transaction may be frequent and belong to any rule. So, it is important for the client to remove all infrequent items from its transaction before further processing. We present privacy preserving protocols for the following sub-tasks that enable privacy preserving collation to obtain item recommendations:

1. **Expunge infrequent items and anonymize:** The task (call it \text{PREPROCESS}) is to remove infrequent items, and anonymize the input transaction of the client. We assume that the initial list of items are given identifiers from the range \([U]\), which are publicly available (hence available to the client).

2. **Private-Collate:** Given a set of consequents of rules, along with their respective weights, privately collate them to produce a list of recommended items using the weights associated with them. For this, the client is given a list of identities, with associated weights (which are homomorphically encrypted) that are obtained from the selection of rules. \( \mathcal{C} \) and \( \mathcal{S} \) can execute a two-party \text{DIST-OBLIVIOUS-SORT} and \text{DIST-MERGE-SORT} (these accomplish 1Note that for two strings \( \text{str}, \text{str}' \), it may be that \( h_r(\text{str} \circ \text{r}') = h_r(\text{str}' \circ \text{r}') \), and \( h_s(\text{str} \circ \text{r}') = h_s(\text{str}' \circ \text{r}') \). Yet the corresponding \( \mathcal{C}_r \)-hashes of \( \text{str} \) and \( \text{str}' \) may not be equal.
(3) **De-anonymize and recommend:** Given a final list of \( k \) anonymized item identities, de-anonymize them to obtain the actual names of the recommended items. For this, \( C \) can fetch their actual identifiers by executing OT \( \mathcal{S} \) (similar to Preprocess), to obtain the true identities of the items to be recommended.

**Private protocol for Approx-GSCS-Query.** Recall that the subroutine Simple-LSH-Query, for Approx-GSCS-Query, is as follows: Server chooses \( l \) random maps, where the \( i^{th} \) map \( func_i \), maps a set \( T \subseteq \mathcal{I} \), represented as a characteristic vector \( v_T \), of length \( |\mathcal{I}| \) to a string \( T_i \) of \( k \) bits. Thus, each antecedent \( p \) of our rules (denoted \( P_1, P_2, \ldots, P_i, \ldots, P_{|\mathcal{I}|} \)) is mapped to \( l \) strings \( p_1, p_2, \ldots, p_i, \ldots, p_l \) of length \( k \) bits each. For an input transaction \( T \), an association rule \( p \rightarrow q \) is selected if and only if any of the \( i \) maps \( func_i(v_T) \) exactly matches \( func_i(p) \).

**Pre-processing the \( D \) rules:** We create an enhanced database \( D_e \) by first choosing \( l \) random strings \( r_1, r_2, r_3, \ldots, r_l \in \{0,1\}^s \), where \( s \) is a security parameter (specified later). We then concatenate the above random strings to the \( l \)-maps as follows: \( r_1 \circ func_1(p_1), r_2 \circ func_2(p_1), \ldots, r_l \circ func_l(p_1) \) for each \( i \in D \). The new database \( D_e \) has \( l \cdot D \) elements, each of which stores all relevant information for the association rule. All strings \( r_i \circ func_i(p_1) \), along with corresponding consequents \( q_i \) in \( D_e \) are entered in \( \mathcal{H} \) defined by Exact-Fetch-Prep.

**Pre-processing the query:** The \( C \) obtains the definition of the \( l \) maps, \( func_i, i \in \{1,\ldots,l\} \), along with the random prefixes \( r_i \), which are declared publicly. It then applies the \( l \) maps on the characteristic vector \( v_T \), corresponding to its input transaction \( T \), and computes \( func_i(v_T) \), from which it prepares \( r_i \circ func_i(v_T) \) for \( i \in \{1,\ldots,l\} \).

**Privately receiving answers to the query:** \( C \) queries \( S \) for existence of each string \( r_i \circ func_i(v_T) \), for \( i = \{1,2,\ldots,l\} \), with the \( S \) possessing enhanced data structure \( \mathcal{H} \) using one of the following approaches:

(1) By executing Exact-Fetch-Query, in which case it reveals the values of \( h_v(r_i \circ func_i(v_T)) \) and \( h_s(r_i \circ func_i(v_T)) \), for \( i \in \{1,\ldots,l\} \) to \( S \).

(2) By executing Private-Exact-Fetch-Query (Lemma \( \mathcal{H} \)), which has the privacy guarantees associated with OT.

**Private protocol for the exact implementation of ALL-ASSOC(\( w, t \)):** We build on primitives from Section 4 to design Protocol 5. For brevity, we discuss the special case when \( t = |\mathcal{I}| \). This protocol makes use of the Exact-Fetch-Query(\( p \)) algorithm as a subroutine. After its execution, client \( C \) gets all association rules with weights \( \geq w \) in a privacy preserving manner, and from these rules, it collates the list of recommended items.

**Protocol 5: Private-Exact-ALL-Assoc(\( w, t = |\mathcal{I}| \))**

| Input: | Client: Transaction \( T \) | Server: Threshold weight \( w \), data structure \( \mathcal{H} \) containing \( D \) association rules |
|-------|-------------------------------|-------------------------------------------------|
| Output: | Client: Set of recommended items \( I_{rec} \) |

1. \( \mathcal{A}[1..l] \leftarrow \text{Call Exact-Fetch-Query (Algorithm 3)} \) \( l \) times
2. \( C \leftarrow \mathcal{A}[1..l] \rightarrow S : \text{Execute DIST-MERGE-SORT (a private sorting protocol) based on whether weight} \geq w. \)
3. \( C \leftarrow \mathcal{L} \rightarrow S \); where \( \mathcal{L} \) is the list of sorted association rules
4. \( C \) collates the consequents of rules in \( \mathcal{L} \) and calculates \( I_{rec} \)
| Parameter                        | Symbol | Default value | Range   |
|---------------------------------|--------|---------------|---------|
| RSA modulus size               | \( N \) | 1024          | \( \{1024, 2048\} \) |
| Number of rules                 | \( D \) | \( 10^4 \) | \( 10^4 - 10^6 \) |
| TOP-Assoc output size           | \( k \) | 3            | 3 - 20  |
| TOP-Assoc length                | \( t \) | 3            | 3 - 10  |
| Query size                      | \(|T|\) | 5            | 5 - 20  |

Table 1: Evaluation Metrics

| \( D \) | \(|T|\) | \( t \) | Time | \( D \) | \(|T|\) | \( t \) | Time | \( D \) | \(|T|\) | \( t \) | Time |
|---------|--------|--------|------|---------|--------|--------|------|---------|--------|--------|------|
| 100K    | 10     | 15     | 20   | 5       | 3      | 58     | 58   | 5       | 3      | 229    | 744  |
|         |        |        |      | 5       | 3      | 273    | 931  |
|         | 10     | 15     | 20   | 5       | 3      | 278    | 1279 |
|         |        |        |      | 5       | 3      | 232    | 1413 |
|         | 15     |        |      | 5       | 3      | 212    | 1911 |
|         |        |        |      | 5       | 5      | 213    | 2009 |
|         | 20     |        |      | 5       | 3      | 263    | 2615 |
|         |        |        |      | 5       | 3      | 229    | 2957 |
| 500K    | 10     |        |      | 5       | 3      | 328    |      |
|         |        |        |      | 5       | 3      | 436    |      |
|         | 15     |        |      | 5       | 3      | 3678   |      |
|         |        |        |      | 5       | 5      | 525    |      |
|         | 20     |        |      | 5       | 3      | 3999   |      |
|         |        |        |      | 5       | 5      | 987    |      |

Table 2: Execution times (in milliseconds) incurred by the Exact-ALL-ASSOC \((w, t)\) algorithm for fetching applicable rules. Symbols \( K \) and \( M \) denotes values \(10^3\) and \(10^6\) respectively.

The private versions of the exact implementations of TOP-ASSOC \((k, w, t)\), MAX-ASSOC \((f)\), and ANY-ASSOC \((k, w, t)\) can be designed in a similar manner.

### 6 Experimental evaluation

Our empirical evaluations illustrate that although the computational requirements generally increase, privacy properties can still be guaranteed at roughly the same time scale as the non-private counterparts for reasonably sized problem instances.

**Experimental setup and evaluation metrics.** Experiments were conducted on a laptop equipped with a 2.5 GHz Core i5 processor and 16 gigabyte of memory running Windows 7. All algorithms were implemented in Java version 8 update 60 with allocated heap space of 8 gigabyte. We explored the parameters listed in Table 1 over corresponding ranges to evaluate our algorithms and their private versions. RSA modulus size \( N \) is the key size used in the underlying crypto-system. Increasing \( N \) causes significant reduction of performance while increasing security. To assess this trade-off, we ran experiments using both RSA1024 and RSA2048. All the experiments were executed 1000 times to compute the amortized execution times.

**Evaluating the exact and approximate algorithms.** First, an exact implementation of the ALL-ASSOC \((w, t)\) criterion is evaluated. This criterion was picked because the size of the list of rules output by the exact algorithm is the larger than the outputs of the other criteria. Further, the computational burden imposed by parameter \( k \) for any \( k < D \) is negligible in terms of the total processing time. We generated synthetic datasets and evaluated our implementation, whose median processing times are listed in Table 2. As can be inferred, even when the number of association rules is very large (for instance, see the entry corresponding to \( D \) equal to 1 million), our implementation is observed to be very efficient.

Second, we evaluated APPROX-GSCS-QUERY \((T, DS)\) on two real world transaction datasets: (a) Retail [25], and (b) Accidents [26]. The retail dataset consists of market basket data collected from an anonymous Belgian retail store for approximately 5 months during the period
1999-2000. The number of transactions is 88163 and the number of items is 16470. We use SPMF’s [27] implementation of FP Growth algorithm (setting minimum support and minimum confidence values to 0.001 and 0.01 respectively) to get 16147 association rules. The Accidents dataset consists of traffic accidents during the period 1991-2000 in Flanders, Belgium. The number of transactions in this dataset is 340,184. The attributes capture the circumstances in which the accidents have occurred. The total number of attributes (corresponding to I) is 572. Again, we use SPMF’s [27] implementation of FP Growth algorithm (setting minimum support and minimum confidence values to 0.5 and 0.6 respectively) to get 334566 association rules. Column $T_q$ of table 3 lists the median processing times for a collection of predefined query transactions for these two datasets (the thresholding parameter for antecedent length, $t$, was varied between 1 to 5). $T_q$ and $T_o$ are processing times without and with privacy and $D_o$ is the reduced size of the set of rules considered for private fetching. As can be inferred, these processing times are competitive to the numbers in Table 2 when compared to datasets of similar sizes. Columns $A_{10}$, $A_{16}$ and $A_{32}$ provide accuracies of APPROX-GSCS-QUERY with hash lengths ($K \cdot L$, see Section 3) set to 10, 16 and 32 bits respectively, averaged over 1000 queries of length 3. When the length is only 10 bits, APPROX-GSCS-QUERY suffers from low accuracy as irrelevant association rules fall under same bucket. We omit more extensive results on the approximation quality for brevity (see [10, 9] for performances of similar approximation schemes).

### Evaluating the timing overhead due to privacy preservation.

Table 3 documents the timing overhead introduced by a single 1-n oblivious transfer, which is used to make the exact implementation privacy preserving. The number of rules ($D$) was varied from 1 thousand to 10 thousand and the RSA modulus was varied between 1024 and 2048. Our implementation of the oblivious transfer protocol is single threaded and is based on [28] (a multi-threaded faster implementation can be found in [24], which can be used as a plug-in module to improve overhead time by a factor of magnitude or higher). From the table we can infer that for moderate sized databases, private fetching of applicable rules is competitive and practical. For instance, to fetch applicable rules from a database with 10^4 rules, the median time taken is ~ 40 seconds for a query of size 5 (RSA modulus set to 1024). Practicality is further supported by the fact that in client server settings applicable for most cloud based applications/world wide web, multiple servers will be handling multiple query requests.

Second, the timing overheads incurred by the privacy preserving counterpart of APPROX-GSCS-QUERY for the real datasets is shown in Table 3 (column $T_o$). We choose the RSA...
modulus value to be 1024 here. Although the processing times are now multiple orders of magnitude compared to vanilla processing times (column $T_q$), they are still practical and manageable for an e-commerce setting (again, due to the fact that in practice multiple servers service multiple queries). These times are also comparable to similar sized datasets benchmarked in Table 4. Thus, our solutions and their private versions are very competitive in fetching applicable association rules. Note that we have excluded the processing times incurred for the Collate step because the processing times required for getting recommended items from selected applicable rules were much lower than those for the Fetch step.

7 Conclusion

Our work proposes a rich set of methods for selection and application of association rules for recommendations that have strong theoretical basis as well as pragmatic grounding. The ability to reuse association rules that are frequently used in the industry to bootstrap scalable and privacy-aware recommender systems makes our solutions very attractive to practitioners. Our experiments further highlight the practicality of achieving privacy preserving recommendations for moderate to large-scale e-commerce applications.

References

[1] C. Duhigg, “How companies learn your secrets,” Feb 2012.

[2] G. Lubin, “The incredible story of how target exposed a teen girl’s pregnancy,” Feb 2012.

[3] G. J. Udo, “Privacy and security concerns as major barriers for e-commerce: a survey study,” *Information Management & Computer Security*, vol. 9, no. 4, pp. 165–174, 2001.

[4] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487–499.

[5] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Analysis of recommendation algorithms for e-commerce,” in *Proceedings of the 2nd ACM Conference on Electronic Commerce*. ACM, 2000, pp. 158–167.

[6] ——, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th International Conference on World Wide Web*. ACM, 2001, pp. 285–295.

[7] A. Shrivastava and P. Li, “Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS),” in *Advances in Neural Information Processing Systems*, 2014, pp. 2321–2329.

[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *Proceedings of the 20th Annual Symposium on Computational Geometry*, 2004, pp. 253–262.

[9] B. Neyshabur and N. Srebro, “On symmetric and asymmetric LSHs for inner product search,” in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 1926–1934.
[10] A. Shrivastava and P. Li, “Asymmetric Minwise hashing for indexing binary inner products and set containment,” in Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 981–991.

[11] D. Li, Q. Lv, H. Xia, L. Shang, T. Lu, and N. Gu, “Pistis: a privacy-preserving content recommender system for online social communities,” in Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, 2011, pp. 79–86.

[12] F. McSherry and I. Mironov, “Differentially private recommender systems: building privacy into the Net,” in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 627–636.

[13] S. Zhang, J. Ford, and F. Makedon, “A privacy-preserving collaborative filtering scheme with two-way communication,” in Proceedings of the 7th ACM Conference on Electronic Commerce, 2006, pp. 316–323.

[14] D. Agrawal and C. C. Aggarwal, “On the design and quantification of privacy preserving data mining algorithms,” in Proceedings of the 20th ACM Symposium on Principles of Database Systems, 2001, pp. 247–255.

[15] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci, “Enhancing privacy and preserving accuracy of a distributed collaborative filtering,” in Proceedings of the 2007 ACM Conference on Recommender Systems, 2007, pp. 9–16.

[16] H. Polat and W. Du, “SVD-based collaborative filtering with privacy,” in Proceedings of the 2005 ACM Symposium on Applied Computing, 2005, pp. 791–795.

[17] J. Canny, “Collaborative filtering with privacy via factor analysis,” in Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2002, pp. 238–245.

[18] E. Aïmeur, G. Brassard, J. M. Fernandez, and F. S. M. Onana, “Alambic: a privacy-preserving recommender system for electronic commerce,” International Journal of Information Security, vol. 7, no. 5, pp. 307–334, 2008.

[19] J. Vaidya and C. Clifton, “Privacy preserving association rule mining in vertically partitioned data,” in Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 639–644.

[20] S. J. Rizvi and J. R. Haritsa, “Maintaining data privacy in association rule mining,” in Proceedings of the 28th International Conference on Very Large Data Bases, 2002, pp. 682–693.

[21] M. L. Fredman, J. Komlós, and E. Szemerédi, “Storing a sparse table with $O(1)$ worst case access time,” Journal of the ACM, vol. 31, no. 3, pp. 538–544, 1984.

[22] J. L. Carter and M. N. Wegman, “Universal classes of hash functions,” in Proceedings of the 9th Annual ACM Symposium on Theory of Computing, 1977, pp. 106–112.

[23] R. L. Rivest, “RFC 1321: The MD5 message-digest algorithm,” Internet Engineering Task Force, vol. 143, 1992.
[24] E. Unal and E. Savas, “On acceleration and scalability of number theoretic private information retrieval,” *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2015.

[25] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets, “Using Association Rules for Product Assortment Decisions: A Case Study,” in *Knowledge Discovery and Data Mining*, 1999, pp. 254–260.

[26] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof, “Profiling High Frequency Accident Locations Using Association Rules,” in *Proceedings of the 82nd Annual Transportation Research Board*, 2003, p. 18pp.

[27] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu., and V. S. Tseng, “SPMF: a Java Open-Source Pattern Mining Library,” *Journal of Machine Learning Research*, vol. 15, pp. 3389–3393, 2014.

[28] H. Lipmaa, “An oblivious transfer protocol with log-squared communication,” in *Proceedings of the 8th International Conference on Information Security*. Springer-Verlag, 2005, pp. 314–328.