J Intell Inf Syst (2007) 29:253–278
DOI 10.1007/s10844-006-0018-8

Preserving privacy in association rule mining with bloom filters

Ling Qiu · Yingjiu Li · Xintao Wu

Received: 19 May 2005 / Revised: 4 November 2005 /
Accepted: 17 January 2006 / Published online: 27 January 2007

Abstract Privacy preserving association rule mining has been an active research area since recently. To this problem, there have been two different approaches—perturbation based and secure multiparty computation based. One drawback of the perturbation based approach is that it cannot always fully preserve individual's privacy while achieving precision of mining results. The secure multiparty computation based approach works only for distributed environment and needs sophisticated protocols, which constrains its practical usage. In this paper, we propose a new approach for preserving privacy in association rule mining. The main idea is to use keyed Bloom filters to represent transactions as well as data items. The proposed approach can fully preserve privacy while maintaining the precision of mining results. The tradeoff between mining precision and storage requirement is investigated. We also propose δ-folding technique to further reduce the storage requirement without sacrificing mining precision and running time.

Keywords Association rule mining · Bloom filters · Privacy preserving

L. Qiu (✉)
School of Maths, Physics and Information Technology, James Cook University,
Townsville, Queensland 4811, Australia
e-mail: ling.qiu@jcu.edu.au

Y. Li
School of Information Systems, Singapore Management University,
Singapore 178902, Singapore
e-mail: yjli@smu.edu.sg

X. Wu
Department of Computer Science, University of North Carolina at Charlotte,
Charlotte, NC 28223, USA
e-mail: xwu@uncc.edu

1 Introduction

The explosive growth of the Internet and advances in networking technology are pushing application logic and data processing from corporate data centers out to proxy servers which are located at the edge of network. In concert with this computing environment change, concerns about privacy of individual information have been raised widely. When pushing applications to edge servers, it is essential to provide adequate security measures to protect data from not only malicious outsiders but also edge servers. The main reason is that the edge servers may not be fully trusted or they can be penetrated in hacking activities. Previous work in this area has been focused on how to secure traditional database queries. This line of work includes encryption in outsourced databases (Agrawal, Kiernan, Srikant, & Xu, 2004; Hacigumus, Iyer, Li, & Mehrotra, 2002a; Hacigumus, Iyer, & Mehrotra, 2002b, 2004; Iyer, Mehrotra, Mykletun, Tsudik, & Wu, 2004; Mykletun, Narasimha, & Tsudik, 2004) and authentication in edge computing (Pang & Tan, 2004). Little work has been focused on complex applications such as data mining, statistical analysis, and data exploration when the data are pushed to privacy preserving. It is critical to ensure that the overall performance, usability and scalability of those applications are not affected much when security and privacy properties are enforced. In this paper, we will focus on the task of mining association rules in edge computing scenario. Our goal is to achieve the same (or even better) performance as in traditional corporate data centers while fully preserving privacy in data mining process.

Association rule mining has been an active research area since its introduction (Agrawal, Imilienski, & Swami, 1993). Many algorithms have been proposed to improve the performance of mining association rules or frequent itemsets. An interesting direction is the development of techniques that incorporate privacy concerns. One type of these techniques is perturbation based, which perturbs the data to a certain degree before data mining so that the real values of sensitive data are obscured while non-sensitive statistics on the collection of data are preserved. Some recent work (Agrawal & Aggarwal, 2001; Atallah, Bertino, Elmagarmid, Ibrahim, & Verykios, 1999; Dasseni, Verykios, Elmagarmid, & Bertino, 2001; Evfimievski, Srikant, Agrawal, & Gehrke, 2002; Evfimievski, Gehrke, & Srikant, 2003; Oliveira & Zaiane, 2002, 2003a, 2003b; Rizvi & Haritsa, 2002; Saygin, Verykios, & Clifton, 2001) investigates the tradeoff between leakage of private information and accuracy of mining results. One drawback of this approach is that it cannot always fully preserve privacy of data while achieving precision of mining results (Kargupta, Datta, Wang, & Sivakumar, 2003). The second type of these techniques is distributed privacy preserving association rule mining (Kantarcıoğlu & Clifton, 2002; Vaidya & Clifton, 2002) based on secure multiparty computation (Yao, 1986). Though this approach can fully preserve privacy, it works only in distributed environment (it requires two or more parties to collaborate in mining process) and needs sophisticated protocols (secure multiparty computation based), which makes it infeasible for our scenario.

1.1 Assumptions

In this paper, we continue the investigation of preserving privacy in association rule mining. We propose the use of keyed Bloom filters for representing data and run modified association rule mining algorithm over Bloom filters so as to preserve privacy. Bloom filter (Bloom, 1970) is a computationally efficient hash-based probabilistic scheme that can represent a set of objects with minimal memory requirements. It can be used to answer membership queries with zero false negatives and low false positives. Bloom filters have been used by various applications in network systems (Border & Mitzenmacher, 2002; Fan, Cao, Almeida, & Border, 2000) and in database systems (Cohen & Matias, 2003; Li & Ross, 1995; Mullin, 1990). We augment the hash functions used in Bloom filters with a secret key so as to preserve the privacy of data in mining process.

Figure 1 illustrates a client-server-based architecture for our data mining problem. Suppose we have a center data server and many clients. The center data server and clients belong to the same party, whereas the edge servers are the external parties to which we outsource partial data mining tasks. For example, the center data server could be the headquarter (or certain branch such as the IT department) of a company which may be located in New York and clients the other branches which may be located in Singapore, London, Shanghai, and other places over the world. Branches (clients) may send data mining requests to the headquarter (the center data server). If all the mining requests are run at the center data server, the center server may be overloaded which may result in delay of responding to requests and delay of transferring mining results. By outsourcing, we may authorize partial mining tasks to an edge server in London to process the requests by branches nearby (e.g., in London or in Manchester) and thus to avoid overloading and delay.

To preserve privacy, the original data is first transformed to a collection of keyed Bloom filters using a secret key by the center data server. The transformed collection of data is then sent to various edge servers that execute association rule mining requests from clients. A client will receive a collection of Bloom filters that represent frequent itemsets and recover original frequent itemsets with a shared key.

We assume that there is no collusion between the edge servers and clients. Therefore, without knowing the secret key and all the hash functions that map 1s to a Bloom filter, the edge servers are unable to interpret the mining results. We also assume that the primary goal of privacy preserving is to prevent an adversary (e.g., possibly certain edge server) from obtaining sensitive data (e.g., the frequent itemsets or the compositions of transactions) from public data (i.e., the Bloom filters of transactions) while outsourcing mining tasks, as studied in (Kantarcıoğlu, Jin, & Clifton, 2004). In Subsection 2.2, we shall discuss how this goal can be achieved by Bloom filters. However, how difficult it is for an adversary to decipher all hash functions and interpret (by certain cryptographic attack, e.g., brute force attack) all items from the Bloom filters of transactions deserves further study especially in cryptography.

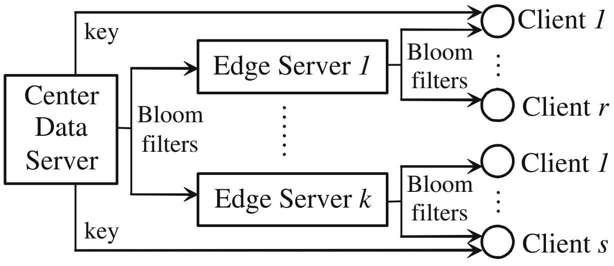


Fig. 1 The client-server-based architecture

It is possible that the requests of data mining tasks may come from customers outside the company. One of the solutions is to add such a customer as a client in the architecture shown in Fig. 1. Another solution is to send the recovered results to this customer. This can protect the mechanism of Bloom filters. However, both solutions may result in exposing of (partial) sensitive data because of potential collusion between such unreliable customers and the edge servers. Therefore, how to protect privacy under collusion is a challenging direction for future study.

1.2 Contributions of the paper

Compared with perturbation based approach, our method can achieve high precision in data mining while fully preserving the privacy of individual data records. It does not require costly cryptographic operations (as mining over encrypted data) or sophisticated protocols (as secure multiparty computation based) to preserve privacy. Moreover, our approach provides flexibility of storage requirement with respect to precision demands in practice.

In brief, our paper has the following contributions:

- We propose a new approach to preserving privacy in association rule mining which is different from both perturbation based and secure multiparty computation based approaches; we also give theoretical analysis on the effectiveness of our approach.
- We define a framework for mining association rules from Bloom filters with adjusted threshold; our experiments show that the mining precision is not sensitive to the threshold adjustment.
- We test the proposed approach rigorously on both synthetic and real datasets; our experiments show that the proposed approach is effective and flexible for practical usage.
- We propose δ-folding technique for saving storage without degrading mining precision and performance. As indicated in our real data experiments, more than a half storage can be saved due to the use of this technique.

1.3 Related work

Agrawal and Srikant (2000) first proposed the development of data mining techniques that incorporate privacy concerns and illustrated a perturbation based approach for decision tree learning. Another class of privacy preserving techniques was investigated intensively (e.g., see Du & Atallah, 2001; Pinkas, 2002, for surveys) and then further studied in Du and Zhan (2002); Lindell and Pinkas (2002) for distributed privacy preserving data mining using secure multiparty computation protocol.

Some effort has been made to address the problem of privacy preservation in association rule mining. For distributed privacy preserving association rule mining, it was considered in Kantarcıoğlu and Clifton (2002); Vaidya and Clifton (2002) the problem over vertical and horizontal partitioned data, respectively. For perturbation (or randomization)based approach, recent research has been focused on the tradeoff between the information leakage and accuracy of mining results. In Atallah et al. (1999); Dasseni et al. (2001), the authors considered the problem of limiting disclosure of sensitive rules, aiming at selectively hiding some frequent itemsets from large databases with as little impact on other, non-sensitive frequent itemsets as possible.

The idea is to modify a given database so that the support of a given set of sensitive rules decreases below a predetermined threshold. Similarly, in Saygin et al. (2001) a method is presented for selectively replacing individual values with unknowns from a database to prevent the discovery of a set of rules,while minimizing the side effects on non-sensitive rules. In Oliveira and Zaiane (2003b), the authors studied the impact of hiding strategies on an original data set by quantifying how much information is preserved after sanitizing the data set. Researchers also studied in Evfimievski et al. (2002); the problem of mining association rules from transactions in which the data has been randomized to preserve the privacy of individual transactions. One problem of this approach is that it may introduce some false association rules.

Related, but not directly relevant to our work, is the research in outsourced databases (Agrawal et al., 2004; Hacigumus et al., 2002a, 2002b; Hacigumus, Iyer, & Mehrotra, 2004; Iyer et al., 2004; Mykletun et al., 2004). Hacigumus et al. (2002a, 2002b); Hacigumus, Iyer, and Mehrotra (2004); Iyer et al. (2004) explored a new paradigm for data management in which a third party service provides hosts database as a service. They explored several encryption techniques in order to process as many queries as possible at the service providers' site, without having to decrypt the data. Recently, Agrawal et al. (2004) presented an order-preserving encryption scheme for numeric data that allows comparison operations to be directly applied on encrypted data. However,encryption is time consuming and it may require auxiliary indices. It is only designed for certain type of queries but not suitable for complex tasks such as association rule mining.

1.4 Organization of the paper

The remaining sections are organized as follows. Section 2 reviews the basics of Bloom filters and formulates our problem. Section 3 presents theoretical analysis for the formulated problem. Section 4 outlines our algorithm for mining frequent itemsets with Bloom filters. Section 5 reports experimental results of both synthetic and real datasets.Section 6 concludes the paper and points out some future directions. Appendix A gives the proofs of the theorems and heuristic that appear in the paper.

2 Problem formulation

In this section, we first review the basics of Bloom filters, and then formulate the problem of preserving privacy in association rule mining with Bloom filters.

2.1 Bloom filter revisited

A Bloom filter is a simple, space-efficient, randomized data structure for representing a set of objects so as to support membership queries.

Definition 2.1 Given an n-element set $S = \{s_1, \ldots, s_n\}$ and k hash functions $h_1, \ldots, h_k$ of range m, the Bloom filter of S, denoted as $B(S)$, is a binary vector of length m that is constructed by the following steps: (i) every bit is initially set to zero; (ii) every element $s \in S$ is hashed into the bit vector through the k hash functions, and the corresponding bits $h_i(s)$ are set[1] to one. A Bloom filter function, denoted as $B(\cdot)$, is a mapping from a set (not necessarily n-element set) to its Bloom filter.

For membership queries, i.e., whether an item $x \in S$, we hash x to the Bloom filter of S (through those hash functions) and check whether all $h_i(x)$ are 1s. If not, then clearly x is not a member of S. If yes, we say x is in S although this could be wrong with some probability.

Definition 2.2 For an element s and a set S, define $s \in_B S$ if s hashes to all 1s in the Bloom filter of S, and $s \notin_B S$ otherwise. The false positive rate of the Bloom filter of S is defined as the probability of $s \in_B S$ while $s \notin S$, or $\Pr(s \in_B S \mid s \notin S)$.

Assume that all hash functions are perfectly random. We have the following

Lemma 2.3 Given an n-element set $S = \{s_1, \ldots, s_n\}$ and its Bloom filter $B(S)$ of length m constructed from k hash functions, the probability for a specific bit in $B(S)$ being 0 is

$$p_0 = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

and the probability for a specific bit being 1 is

$$p_1 = 1 - p_0 \approx 1 - e^{-kn/m}.$$

Then the false positive rate of $B(S)$ is

$$f = p_1^k \approx \left(1 - e^{-kn/m}\right)^k \tag{1}$$

Given n and m, one can minimize the false positive rate f by choosing $k = \frac{m}{n} \ln 2$, in which case $p_0 = p_1 = 0.5$ and $f = 0.5^k = (0.6185)^{m/n}$.

Keyed Bloom Filter To preserve the privacy of Bloom filters, we augment the hash functions h_i with a secret key K. To represent set S, an element $s \in S$ is inserted into Bloom filter B by setting the corresponding bits $h_i(s||K)$ in B to one, where $||$ represents concatenation. Also, to query whether an item $x \in S$, we check whether all $h_i(x||K)$ bits are set to 1. Without knowing the secret key, one is not able to know which set is represented by examining a Bloom filter only. Without further mention we always assume that Bloom filters are constructed with a secret key.

2.2 Our problem

For market basket data, we define each transaction, such as a list of items purchased, as a subset of all possible items.

Definition 2.4 Let $\mathcal{I} = \{I_1, \ldots, I_d\}$ be a set of d boolean variables called items. Let database D be a set of transactions $T_1, T_2, \ldots, T_N$ where each transaction T_i is a set

[1] A location can be set to 1 multiple times, but only the first change has an effect.

of items such that $T_i \subseteq \mathcal{I}$. The support of an itemset S over $\mathcal{I}$, denoted $support(S)$, is defined as the number of the transactions that contain S. The frequency of an itemset S, denoted $freq(S)$, is defined as $support(S)/N$.

Problem 1 *Frequent Itemsets Mining* Given a transaction database $\mathcal{D}$ over $\mathcal{I}$ and a threshold $\tau \in [0, 1]$, find all frequent itemsets $FS \in 2^{\mathcal{I}}$ such that $freq(FS) \geqslant \tau$.

The frequent itemset mining has been a common task in many data mining projects for the past decade. From frequent itemsets, one can easily derive all associate rules (Agrawal & Srikant, 1994) of the kind *if A, then likely B* where A, B are frequent itemsets. The mining of frequent itemsets of association rules has a wide range of applications in many areas, from the analysis of customer preferences to DNA patterns.

Our idea is to transform transaction database to a collection of Bloom filters to preserve the privacy in frequent itemset mining. Each transaction $T_i \in \mathcal{T}$ is transformed to Bloom filter $B(T_i)$ of size m using k hash functions. To preserve the privacy of items I_i, we assume that the mining process is done on the (keyed) Bloom filters $B(I_i)$ of the items rather than on the items themselves.

Problem 2 *Privacy Preserving Frequent Itemsets Mining* Given (i) a collection of Bloom filters$\{B(T_1), \ldots, B(T_N)\}$ for transaction database $\mathcal{D}$ over $\mathcal{I}$, (ii) a set of Bloom filters $\{B(I_1), \ldots, B(I_d)\}$ for items in $\mathcal{I}$, and (iii) a threshold $\tau \in [0, 1]$, find all Bloom filters $B(FS)$ of itemsets $FS \in 2^{\mathcal{I}}$ such that $freq(FS) \geqslant \tau$.

The data mining task can be performed on an edge server where the database is outsourced. The edge server is provided with Bloom filters for performing the data mining task. As mentioned earlier in Subsection 1.1, our primary goal of privacy preserving is to prevent an adversary (possibly an edge server) from obtaining sensitive data from public data (Kantarcıoğlu et al., 2004). This goal can be achieved by Bloom filters which satisfy simultaneously the following three conditions. First, transactions containing different numbers of items are mapped to Bloom filters with the same length. This prevents an adversary from deciphering the compositions of transactions by analyzing the lengths of Bloom filters. Second, Bloom filters support membership queries. This allows authorized external parties (e.g., the edge servers) to carry out data mining tasks with only Bloom filters (i.e., Bloom filters of transactions and candidates of frequent itemsets). Third, without knowing all individual items, it is difficult to identify each item from a transaction Bloom filter by counting the numbers of 1s and 0s. This is because the probability of a bit in a Bloom filter being 1 or 0 is 0.5 (Lemma 2.3) given that the parameters of a Bloom filter satisfy $k = \frac{m}{n} \ln 2$ as we have followed in this paper.

Some transactions may contain only one item. The number of 1s in these Bloom filters are no more than but very close to k. Therefore, the outsourced database may divulge partial individual items. To prevent such divulgence, one of the solutions is to insert several virtual items as white noise to those transactions in which item numbers are smaller than a threshold.

When a client sends mining requests, it has to send candidates of frequent k-itemsets to an edge server at the same time (see Section 4 for detailed mining process). Candidates of frequent 1-itemsets are exactly individual items. To prevent from exposing them to edge servers, it is advisory not to outsource tasks of mining frequent 1-itemsets. There is an alternative way to conceal individual items by extending the usage of secret key discussed in the previous subsection. We first insert several virtual items, denoted by $k_1, k_2, \ldots, k_i$, into all transactions and then outsource them to the edge servers. At client side, each candidate of frequent 1-itemsets is inserted with a virtual item randomly chosen from k_1 to k_i. Thus the edge servers cannot easily identify candidates of frequent 1- and 2-itemsets because both types of candidates look alike. This method can be applied to conceal candidates of frequent 2-, 3-, ..., and k-itemsets. Moreover, this operation can be done before or after the mixing of white noise discussed in the previous paragraph.

3 Analysis

For any given itemset, the support (frequency) learnt from Bloom filters may be greater than its real support (frequency) learnt from original transactions due to the false positive of Bloom filters. We make this clear in the following analysis. By default, we assume that for any itemset, there is a Bloom filter function $B(\cdot)$ which produces binary vector of length m through k hash functions.

Definition 3.1 Given an itemset S and a transaction T_i, define $S \subseteq_B T_i$ if for all items $s \in S$, $s \in_B T_i$, and define $S \nsubseteq_B T_i$ otherwise. The false positive rate for checking S from the Bloom filter of T_i, denoted as f_i, is defined as the probability of $S \subseteq_B T_i$ while $S \nsubseteq T_i$, or $\Pr(S \subseteq_B T_i \mid S \nsubseteq T_i)$.

Sometimes, we also use $B(S) \sqsubseteq B(T_i)$ to represent $S \subseteq_B T_i$. Now we can define the support and frequency that are learnt from Bloom filters.

Definition 3.2 Given a collection of N Bloom filters $\{B(T_1), \ldots, B(T_N)\}$ for transaction database $\mathcal{D}$ over $\mathcal{I}$, the support of an itemset $S \in 2^{\mathcal{I}}$ that is learnt from the collection of filters,denoted as $Bsupport(S)$, is defined as the number of filters $B(T_i)$ that satisfy $S \subseteq_B T_i$. The frequency of S that is learnt from the collection of filters, denoted as $Bfreq(S)$, is defined as $Bsupport(S)/N$.

From Definition 3.1, one can verify the following lemma which indicates that $Bsupport(S)$ can be computed from $B(S)$ without S and that $Bsupport(S) \geqslant support(S)$ due to false positive.

Lemma 3.3 (i) $S \subseteq_B T_i$ iff $B(S) \wedge B(T_i) = B(S)$, where $\wedge$ is bitwise AND. (ii) If $S \subseteq T_i$, then $S \subseteq_B T_i$. (iii) If $S \nsubseteq T_i$, then $S \subseteq_B T_i$ with probability f_i, and $S \nsubseteq_B T_i$ with probability $1 - f_i$.

Theorem 3.4 *Given an itemset S and a transaction T_i, the false positive rate of checking S from the Bloom filter of T_i is*

$$f_i = \left(1 - e^{-kn_i/m}\right)^{||B(S-T_i)||}$$

where $n_i = |T_i|$ is the length of transaction T_i, and $||\cdot||$ indicates the number of 1*s in a binary vector.*

When $S \nsubseteq T_i$, we define a random 0–1 variable $e_i = 1$ if $S \subseteq_B T_i$, and $e_i = 0$ if $S \nsubseteq_B T_i$. The variable indicates whether $S \subseteq_B T_i$. From Lemma 3.3, one can verify that $e_i = 1$ with probability f_i and $e_i = 0$ with probability $1 - f_i$. In other words, e_i represents an independent Bernoulli trial with probability f_i of success. Its mean and variances $E[e_i] = f_i$ and $Var[e_i] = f_i(1 - f_i)$, where we use $E[\cdot]$ and $Var[\cdot]$ to denote the mean and variance of any random variable, respectively.

From the previous discussion, we have

$$Bsupport(S) = support(S) + s_e$$

where $s_e = \sum_{i=1}^{N-support(S)} f_i$ if we assume that the first $N - support(S)$ transactions do not contain S.

Lemma 3.5 Given n random variables $x_1, \ldots, x_n$, we have $E[x_1 + \cdots x_n] = E[x_1] + \cdots + E[x_n]$and $Var[x_1 + \cdots + x_n] = \sum_{i,j=1}^{n} E\Big[\big(x_i - E[x_i]\big)\big(x_j - E[x_j]\big)\Big]$. If $x_1, \ldots, x_n$ are independent, we have $Var[x_1 + \cdots + x_n] = Var[x_1] + \cdots + Var[x_n]$.

From Lemma 3.5, one can derive

$$E[s_e] = \sum_{i=1}^{N-support(S)} f_i = \big(N - support(S)\big)\,\bar{f}$$

$$V[s_e] = \sum_{i=1}^{N-support(S)} f_i(1 - f_i)$$

where $\bar{f} = \frac{1}{N-support(S)} \sum_{i=1}^{N-support(S)} f_i$ is *average false positive rate* for checking itemset S from transactions. By choosing the optimal hash function number $k = \frac{m}{n} \ln 2$, one can estimate $\bar{f} \approx 0.5^{||B(S)||}$ (see Appendix B).

Lemma 3.6 [Chernoff bound (Chernoff, 1952; Hoeffding, 1963)] Let $X_1, \ldots, X_n$ be independent 0–1 random variables with $\Pr(X_i = 1) = p_i$. Let $X = \sum_i X_i$, $\mu = E[X]$ and $p = \mu/n$. Then for $0 \leqslant \lambda < n - \mu$,

$$\Pr(X \geqslant \mu + \lambda) \leqslant e^{n \cdot H_p\left(p + \frac{\lambda}{n}\right)}$$

$$\Pr(X \leqslant \mu - \lambda) \leqslant e^{n \cdot H_{1-p}\left(1 - p + \frac{\lambda}{n}\right)}$$

where $H_p(x) = x \ln\left(\frac{p}{x}\right) + (1 - x) \ln\left(\frac{1-p}{1-x}\right)$ is the relative entropy of x w.r.t. p.

Theorem 3.7 *Let $\mu = E[e_s]$, $min\_sup = \tau N$, and $\delta = |Bsupport(S) - \mu - min\_sup|$. If $Bsupport(S) \geqslant min\_sup + \mu$, then $support(S) \geqslant min\_sup$ with probability at least $1 - \epsilon_1$; otherwise$support(S) < min_sup$ with probability at least $1 - \epsilon_2$, where*

$$\epsilon_1 = e^{\left(N - support(S)\right) \cdot H_{\bar{f}}\left(\bar{f} + \frac{\delta}{N - support(S)}\right)}$$

$$\epsilon_2 = e^{\left(N - support(S)\right) \cdot H_{1-\bar{f}}\left(1 - \bar{f} + \frac{\delta}{N - support(S)}\right)}$$

Note that Chernoff bounds are not always very tight. Exact probabilities or tighter bounds can be computed from the binomial distribution or its normal approximation in practice.

Solution to problem 2 Given a Bloom filter of an itemset S and a collection of Bloom filters of transactions $\{B(T_i)\}$, one can easily compute $Bsupport(S)$ by checking $B(S)$ against $\{B(T_i)\}$. Let $min\_sup' = min\_sup + \mu$ be *revised threshold*. We consider S "frequent" if $Bsupport(S) \geqslant min\_sup'$, and output $B(S)$ for problem 2. According to Theorem 3.7, ϵ_1 is the upper bound of the probability that S is actually not frequent. We thus call ϵ_1 *false frequent* rate. Similarly, we call ϵ_2 *false infrequent* rate which is the upper bound of the probability that an "infrequent" itemset (i.e., $Bsupport(S) < min\_sup'$) is actually frequent. For convenience, both ϵ_1 and ϵ_2 are called *error rates*. The error rates increase with $\bar{f}$ and decrease with δ.

In practice, one can choose smaller $\bar{f}$ to decrease both error rates. One can also change min_sup' to tradeoff between the error rates. If the false infrequent rate is of main concern, one can lower the revised threshold by δ such that the false infrequent rate is less than a predetermined threshold ϵ, where δ is the solution to $\epsilon_2 = \epsilon$. In this case, fewer frequent itemsets will be missed as the false infrequent rate is less than ϵ. On the other hand, however, there could be more infrequent itemsets that are included in the mining result.

4 Our method

A framework of our method for mining frequent itemsets with Bloom filters is shown in Algorithm 1. Algorithm 1 can be divided into three phases: *counting phase* (lines 3–5), *pruning phase* (lines 6–10), and *candidates generating phase* (lines 11–12). In the following, we present our method phase by phase. In particular, we study various techniques so as to improve the efficiency of our algorithm, and study how to extend our method so that it can be more flexible in practice.

4.1 Counting phase

In the counting phase (lines 3–5 in Algorithm 1), each candidate filter is checked against all transaction filters and the candidate's count is updated. A straightforward method is to check each combination of candidate filter and transaction filter in a brute force manner. However, this method may not be efficient.

To improve the efficiency, we organize the candidate Bloom filters in a tree hierarchy and use every q bits to partition them at different levels, where q is a parameter. For example, at the root level, the partition leads to 2^q child nodes; the

Algorithm 1 Mining frequent itemsets with Bloom filters

```
1: $C_1 = \{B(I_1), \ldots, B(I_d)\}$                 // $B(I_i)$ is the Bloom filter of item $I_i$
2: for ($\ell = 1$; $C_\ell \neq \varnothing$; $\ell$++) do
3:     for each $B(S) \in C_\ell$ and each transaction filter $B(T_i)$ do
4:         if $B(S) \sqsubseteq B(T_i)$ then Bsupport(S)++   // $B(S) \sqsubseteq B(T_i)$ iff $B(S) \wedge B(T_i) = B(S)$
5:     end for
6:     for each $B(S) \in C_\ell$ do
7:         $\bar{f} = 0.5^{||B(S)||}$
8:         $min\_sup' = min\_sup + \mu$        // $\mu = \big(N - Bsupport(S)\big)\frac{\bar{f}}{1-\bar{f}}$, see below
9:         if $Bsupport(S) < min\_sup'$ then delete $B(S)$ from $C_\ell$
10:    end for
11:    $F_\ell = C_\ell$      // $F_\ell$ is the collection of Bloom filters of all "frequent" itemsets with
                              length $\ell$
12:    $C_{\ell+1} = can\_gen(F_\ell)$    // generate filters of candidate itemsets for the next round
13: end for
14: Answer = $\bigcup_\ell F_\ell$                    // all filters of frequent itemsets
```

Bloom filters in each node share the same first q bits. A node splits if it contains more than c Bloom filters, where c is another parameter. At the end of partition, each leaf node contains limited number of Bloom filters, while each non-leaf node (except the root) is associated with a q-bit segment by which the node splits.

Because of the randomness of keyed hash functions, the distribution of Bloom filters is uniform, which implies that the tree is well balanced. Therefore, an L-level tree can be used to index up to $c \cdot 2^{qL}$ candidate Bloom filters. Given $q = 5$ and $c = 20$, for example, a 4-level tree can be used to index 20M Bloom filters.

Heuristic 4.1 Let s be the q-bit segment associated with a non-leaf note and $B(T_i)$ be a transaction Bloom filter. If any bit in s is 1 while the corresponding bit in $B(T_i)$ is 0, then no Bloom filter in the subtree from the non-leaf node needs to be checked in the counting phase.

In the counting phase, we traverse the tree to compare each transaction filter with candidate Bloom filters (stored in leaf nodes) and update their counts. According to the above heuristic, we can skip many subtrees in the process.

An alternative way to do this is to organize transaction filters in a tree structure and update their counts appropriately while traversing the tree for each candidate filter.

4.2 Pruning phase

In the pruning phase (lines 6–10 in Algorithm 1), any Bloom filter is eliminated from the candidate set if its count (i.e., Bsupport) is less than the revised threshold. According to Section 3, the revised threshold is $min\_sup' = min\_sup + \mu$, where $\mu = \big(N - support(S)\big)\bar{f}$. Since $support(S)$ is unknown in data mining, we solve it by replacing $support(S)$ with $Bsupport(S) - \mu$, and derive $\mu = \big(N - Bsupport(S)\big)\frac{\bar{f}}{1-\bar{f}}$. The revised threshold is computed for each candidate Bloom filter.

4.3 Candidates generating phase

With counting phase and pruning phase alone, our algorithm can discover all Bloom filters of frequent itemsets from *any* given set of Bloom filters (not necessarily Bloom filters of single items). This can be done with one step interaction between client and server—the client generates a set of candidate Bloom filters C_1 and sends it to the server; after data mining, the server returns the result F_1 to the client. The client can easily recover all frequent itemsets from those Bloom filters. This one step interaction is useful in some application scenarios. As mentioned in Subsection 2.2, with C_1 (i.e., Bloom filters of individual items) an edge server may decipher partial sensitive data (e.g., the compositions of transactions). Therefore, for privacy preserving concern, it is advisory not to outsource frequent 1-itemset mining tasks. Also, an alternative choice of concealing C_1 has been discussed in Subsection 2.2.

To support interactive data mining and discover all frequent itemsets as required in Problem 2, a multiple step interaction is conducted between client and server. The client provides the server with a set of candidate filters C_ℓ; after the server sends back the mining result F_ℓ, the client generates another set $C_{\ell+1}$ of candidate filters from F_ℓ and sends it to the server for data mining. This process can be done repetitively.

The candidate generation $C_{\ell+1} = can\_gen(F_\ell)$ at client side is conducted in the following steps for privacy reason. First the Bloom filters in F_ℓ are transferred back to itemsets. This can be done by maintaining and following a one-to-one mapping between each Bloom filter in C_ℓ and its corresponding itemset (note that F_ℓ is a subset of C_ℓ). From the collection of all itemsets, the client generates a new set of candidate itemsets using *apriori_gen* as proposed in Agrawal and Srikant (1994). The basic idea of *apriori_gen* is that a candidate itemset of length $\ell + 1$ is generated only if all its subsets of length ℓ appear in the collection of itemsets. The client may also edit the set of candidate itemsets according to application requirements and constraints. Finally, the client transfers the candidate itemsets to Bloom filters and sends them back (in $C_{\ell+1}$) to the server. All of our experiments presented in the next section are based on this scenario.

Due to false positive, an itemset S transferred from Bloom filter $B(S)$ could be a superset of the real frequent itemset[2] that corresponds to the same Bloom filter $B(S)$ (one can choose longer Bloom filters so as to make S close enough to the set of real frequent itemsets). Fortunately, S is *not* a proper subset of the set of real frequent itemsets. Therefore, no real frequent itemsets are missed out in the candidate generation for the next round.

Another choice to perform candidate generation is at server side. However, the server has no secret key to perform the hash functions, so it cannot transfer back and forth between Bloom filters and itemsets. A possible solution is to use $C_{\ell+1} = \{B(S_1) \vee B(S_2) : B(S_1), B(S_2) \in F_\ell\}$ as candidate set for the next round,where $\vee$ represents bitwise OR. It is easy to verify that $B(S_1) \vee B(S_2)$ is the Bloom filter of itemset $S_1 \cup S_2$; therefore, this solution generates all Bloom filters of itemsets that are unions of any two frequent itemsets (clearly, no frequent itemset is missed in this process). The disadvantage of this solution is that the server cannot exploit Apriori property (using *apriori_gen*) at itemset level.

[2]By real frequent itemset we mean its support is no less than *min_sup*.

4.4 Extension

The method we discussed above is applicable to the case in which all transactions are represented by Bloom filters with the same length (e.g., all are 640-bit). It can be extended to the case in which transactions are represented by Bloom filters with several different lengths.

In some applications, the dataset contains both big size transactions (e.g., containing 200 items) and small size transactions (e.g., containing two or three items). To save storage space, we use long Bloom filters (e.g., 640-bit) to represent big size transactions and short Bloom filters (e.g., 64-bit) to represent small size transactions. Thus the Bloom filters of transactions are classified into two groups based on their lengths. Note that only the counting phase needs to be revised during the mining process. We need to execute counting operations for a small group of Bloom filters (e.g., long Bloom filters), then accumulate the results for the other groups of Bloom filters, and finally execute pruning phase and candidate generating phase.

In order to reduce the communication bandwidth between edge servers and clients, we propose δ-folding technique so that edge servers can generate candidate Bloom filters of different lengths by themselves. The details are given in the next section. Our experiments on real data show that, the extended method achieves not only significantly high rates of storage saving (over 75% in the best case), but also the same mining precision as well as comparable performance.

5 Experiments

To assess the relative performance of our Bloom filter method (BF for short) against Apriori method for discovering frequent itemsets, we conduct experiments on both synthetic data and real data. All the experiments are run on an IBM ThinkPad T40 laptop computer with a Pentium M processor (CPU) clock rate of 1.5 GHz, 1.0 GB of RAM and 40 GB hard disk, running Microsoft Windows XP with SP2.

Let F be the set of frequent itemsets obtained by Apriori and F' by BF. Define $E_p = F' \setminus F$ be the set of positive errors, and $E_n = F \setminus F'$ be the set of negative errors. In other words, E_p contains the item sets overly found by BF, whereas E_n contains the itemsets missed out by BF as compared with the mining results of Apriori.

As aforementioned, our BF method may incur some unexpected itemsets or miss out some itemsets in the mining results. In what follows, we compare our results with the correct results obtained by Apriori.

5.1 On synthetic data

In synthetic data experiments, we investigate the following aspects: (1) mining precision, (2) scalability in terms of running time, and (3) flexibility of storage requirement.

We generate synthetic data using the transaction generator designed by IBM Quest project (Agrawal & Srikant, 1994). A synthetic dataset contains 100K to 1M transactions; each transaction is generated from a set of 1,000 frequent itemsets. The size of each frequent itemset is picked from Poisson distribution with mean 4.

Table 1 Characteristics of synthetic datasets

Dataset	N_t	Distinct items	T_m	T_a	Numbers (and percentage) of transactions with size			
					1–10	11–20	21–30	> 30
Syn-1	100K	1,000	31	10.01	58,272 (58.3%)	41,586 (41.6%)	141 (0.1%)	1 (0%)
Syn-2	250K	1,000	27	10.00	145,768 (58.3%)	103,826 (41.5%)	406 (0.2%)	0 (0%)
Syn-3	500K	1,000	30	10.00	291,529 (58.3%)	207,704 (41.5%)	767 (0.2%)	0 (0%)
Syn-4	750K	1,000	28	10.00	437,372 (58.3%)	311,431 (41.5%)	1,197 (0.2%)	0 (0%)
Syn-5	1M	1,000	29	10.00	583,529 (58.3%)	414,912 (41.5%)	1,559 (0.2%)	0 (0%)

There are totally 1,000 distinct items. The size of each transaction is picked from Poisson distribution with mean 10. We use *five* sets of synthetic data, namely Syn-1, Syn-2, ..., Syn-5. Table 1 shows the characteristics of the synthetic datasets, where N_t denotes the number of transactions in the dataset, T_m and T_a denote the maximum size and average size of transactions, respectively.

Recall that adjusted threshold $min\_sup' = min\_sup + \mu$ is used in data mining process. To be more flexible, we add a coefficient α before μ. Thus the adjustment of minimum support becomes $min\_sup' = min\_sup + \alpha\mu$. From our experimental results, we observe no negative errors for $\alpha = 0$; this is consistent with our theoretical analysis in the previous sections. There is a tradeoff between the numbers of positive and negative errors. In our experiments, we set $0 \leqslant \alpha \leqslant 1$ so as to avoid too many negative errors.

5.1.1 Precision

We examine the precision of BF method using sythetic dataset Syn-1. Recall that $k = \frac{m}{n} \ln 2$ (Lemma 2.3), where k denotes the number of hash functions, m the length of Bloom filter, and n the number of elements in a set. We choose $m = 320$, $n = 10$ (thus $k = 22$).

Table 2 shows the performance in terms of the number of frequent ℓ-itemsets discovered by Apriori and the negative and positive errors of BF. The minimum support varies from 0.25 to 1.50%. In the table, the negative and positive errors are delimited by a comma. Symbol "–" means zero (i.e., no frequent ℓ-itemsets are found).

We use positive (negative resp.) error rate to measure the relative precision. The error rate is defined as follows:

$$\frac{\text{positive errors } |E_p| \text{ (negative errors } |E_n| \text{ resp.)}}{\text{number of frequent itemsets discovered by Apriori}} \times 100\%.$$

From Table 2, we have the following findings:

(1) There are no negative errors at $\alpha = 0$, while a few negative errors (less than 3) occur in *five* cases where $\alpha = 1$.
(2) We achieve high precision of mining results. The positive error rates are no more than 3%.
(3) The adjustment of minimum support threshold slightly affects the mining precision. The positive error at $\alpha = 0$ is slightly greater than that at $\alpha = 1$.

Table 2 (negative, positive) errors where $m = 320$, $n = 10$ and $k = 22$

Min-support	Method	Number of frequent ℓ-itemsets obtained by Apriori, and (negative, positive) errors by BF								
		1	2	3	4	5	6	7	8	Total
	Apriori	781	1,893	1,574	1,047	522	182	47	6	6,052
0.25%	BF $\alpha = 0$	0, 22	0, 12	0, 1	0, 0	0, 5	0, 0	0, 0	0, 0	0, 40
	BF $\alpha = 1$	0, 22	3, 11	3, 0	0, 0	0, 5	0, 0	0, 0	0, 0	6, 38
	Apriori	639	746	492	193	40	2	–	–	2,112
0.50%	BF $\alpha = 0$	0, 17	0, 6	0, 0	0, 1	0, 0	0, 0	–	–	0, 24
	BF $\alpha = 1$	0, 17	2, 5	1, 0	0, 1	0, 0	0, 0	–	–	3, 23
	Apriori	518	309	136	45	9	1	–	–	1,018
0.75%	BF $\alpha = 0$	0, 16	0, 3	0, 0	0, 0	0, 0	0, 0	–	–	0, 19
	BF $\alpha = 1$	0, 15	2, 2	0, 0	0, 0	0, 0	0, 0	–	–	2, 17
	Apriori	396	114	42	13	2	–	–	–	567
1.00%	BF $\alpha = 0$	0, 13	0, 1	0, 0	0, 0	0, 0	–	–	–	0, 14
	BF $\alpha = 1$	0, 12	0, 1	0, 0	0, 0	0, 0	–	–	–	0, 13
	Apriori	225	28	20	4	–	–	–	–	277
1.50%	BF $\alpha = 0$	0, 10	0, 0	0, 0	0, 0	–	–	–	–	0, 10
	BF $\alpha = 1$	0, 10	0, 0	0, 0	0, 0	–	–	–	–	0, 10

The occurrence of negative errors at $\alpha = 1$ is caused by the revision of minimum support threshold. According to the third finding in the above, we can simplify our method by setting $\alpha = 0$ in the following experiments.This guarantees no missing of frequent itemsets, and only incurs a small number of positive errors.

5.1.2 Scalability

We examine the scalability of our method in terms of precision and running time. All *five* datasets given in Table 1 are used in this experiment. We set minimum support

Table 3 Positive errors where $\alpha = 0$ and min-support = 0.75%

Dataset	Transactions	Number of frequent ℓ-itemsets obtained by Apriori, and positive errors by BF							
		1	2	3	4	5	6	7	Total
Syn-1	100K	$\frac{16}{518}$	$\frac{3}{309}$	$\frac{0}{136}$	$\frac{0}{45}$	$\frac{0}{9}$	$\frac{0}{1}$	–	$\frac{19}{1,018}$
Syn-2	250K	$\frac{12}{532}$	$\frac{2}{313}$	$\frac{0}{147}$	$\frac{0}{70}$	$\frac{0}{29}$	$\frac{0}{8}$	$\frac{0}{1}$	$\frac{14}{1,100}$
Syn-3	500K	$\frac{11}{540}$	$\frac{1}{278}$	$\frac{0}{115}$	$\frac{0}{39}$	$\frac{0}{6}$	–	–	$\frac{12}{978}$
Syn-4	750K	$\frac{14}{515}$	$\frac{0}{319}$	$\frac{0}{145}$	$\frac{0}{43}$	$\frac{0}{5}$	–	–	$\frac{14}{1,027}$
Syn-5	1M	$\frac{16}{522}$	$\frac{1}{287}$	$\frac{1}{140}$	$\frac{0}{56}$	$\frac{0}{19}$	$\frac{0}{3}$	–	$\frac{18}{1,027}$

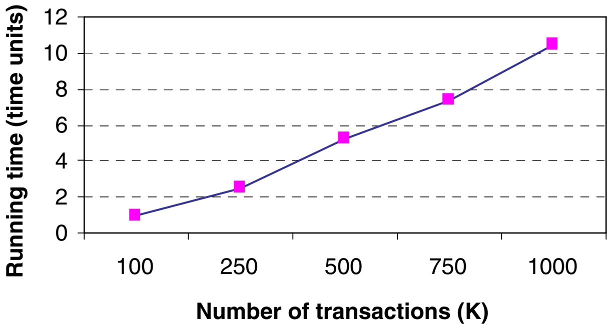


Fig. 2 Trend of running time

to 0.75%, let $\alpha = 0$, and retain other parameters (i.e., $m = 320$, $n = 10$, and $k = 22$). The experimental results are shown in Table 3.

In each cell of Table 3, the denominator denotes the number of frequent ℓ-itemsets obtained by Apriori, whereas the numerator denotes positive errors by BF method.

The table shows that the positive error rate is no more than 2%. One may claim that the precision of mining is stable when we scale up the number of transactions.

The reason of this stability is that all *five* datasets are consistently generated by the same generator with the same set of parameters. The parameters of Bloom filter (i.e., n, m, and k) are optimally chosen such that the errors can be minimized.

Figure 2 illustrates the relative running time w.r.t. the transaction number which corresponds to each synthetic dataset. It shows a linear trend of running time when we scale up the number of transactions.

The running time is in the order of tens of minutes, which is nearly 70 times of that by Apriori. The reason is that, in the counting phase (lines 3–5 in Algorithm 1), a transaction Bloom filter has to be compared with all candidates at certain level of the candidate hash-tree (the candidates to be compared each time is as many as $c \cdot 2^q$ as mentioned in Subsection 4.1); whereas in Apriori, a transaction is directly mapped by a hash function such that it only needs to be compared with several candidates at certain level of the candidate hash-tree.

Compared with Apriori, our BF method sacrifices some running time so as to fully preserve data privacy. Nonetheless,the mining time is still acceptable. Note that we use a laptop in our experiments; in real application scenarios, the data mining task would be performed by edge servers which are much more powerful than our laptop.

5.1.3 Flexibility of storage requirement

Now we conduct experiments to investigate the relationship between error rate and length of Bloom filter. In this experiment, we use dataset Syn-1, and let minimum support be 0.75% and $\alpha = 0$. Since the average size of transactions n is fixed to 10, we vary the Bloom filter length m from 224 bits to 416 bits, and accordingly the Hash function number k from 16 to 29 given the relationship $k = \frac{m}{n} \ln 2$.

Table 4 shows our experimental results on positive errors. In the table, L_{BF} indicates the length of Bloom filters measured in bits, and N_H indicates the corresponding number of hash functions. For relative comparison, we also list the numbers of frequent ℓ-itemsets obtained by Apriori in the third row of the table.

Table 4 Positive errors for dataset Syn-1 where $\alpha = 0$ and min-support= 0.75%

L_{BF} (bits)	N_H	Frequent ℓ-itemsets						
		1	2	3	4	5	6	Total
Apriori		518	309	136	45	9	1	1,018
224	16	124	72	4	0	0	0	200
256	18	78	13	0	0	0	0	91
288	20	38	5	0	0	0	0	43
320	22	16	3	0	0	0	0	19
352	24	10	2	0	0	0	0	12
384	27	5	0	0	0	0	0	5
416	29	2	0	0	0	0	0	2

Table 4 shows that the positive error decreases with Bloom filter length: 200 errors (error rate 19.6%) for 224 bits Bloom filters, 19 errors (error rate is 1.9%) for 320 bits Bloom filters, and 2 errors (error rate $\sim$ 0%) for 320 bits Bloom filters.

The above results demonstrate that more storage space is required to achieve low error rates. A tradeoff between Bloom filter length and error rate can be explored in practice.

An interesting finding is that there are few errors for frequent itemsets whose lengths are greater than two. If clients are interested in mining such frequent itemsets or verifying whether some such itemsets are frequent, they can use relatively short Bloom filters as long as the demand on mining precision is met. By doing so, one can further save storage space.

Discussion. Our BF method is flexible in terms of storage requirement. It provides a tradeoff between storage requirement and the mining errors that we can tolerate in real applications. Normally, each transaction is stored as a series of items represented by integers. In our experiments, the mean size of transactions is ten. Thus on average a transaction requires 40 bytes if an integer is stored by 4 bytes. With our BF method, a transaction is represented by one Bloom filter regardless transaction size. Therefore, if 256-bit Bloom filters are used, one can save 20% storage space with error rate 9.0%; whereas 288-bit Bloom filters are used, 10% storage space is saved with error rate 4.2%. However, this does not mean that one can eliminate errors by increasing the length of Bloom filters. This issue will be addressed in the next section.

5.2 On real data

In real data experiments, we investigate: (1) the mining precision and the limitation of our original BF method, (2)the improved method for increasing mining precision, and (3) the technique for saving more storage space.

The real datasets we adopt are BMS-POS, BMS-WebView-1, and BMS-WebView-2.[3] The dataset BMS-POS contains several years worth of point-of-sale data from a large electronics retailer, whereas the datasets BMS-WebView-1 and

[3]They are downloadable at http://www.ecn.purdue.edu/KDDCUP.

Table 5 Characteristics of real datasets

Dataset	Distinct items	Max-size	Aver-size	Numbers (percentage) of transactions with size				
		T_m	T_a	$\geqslant 1$	1–10	11–20	21–30	> 30
BMS-POS	1,657	164	6.53	515,597	421,113 81.7%	71,617 13.0%	16,544 3.2%	6,323 1.2%
BMS-WebView-1	497	267	2.51	59,602	58,239 97.7%	1,043 1.7%	186 0.3%	134 0.2%
BMS-WebView-2	3,340	161	4.62	77,512	69,678 89.9%	5,840 7.5%	1,035 1.3%	689 0.9%

BMS-WebView-2 contain several months worth of click stream data from two e-commerce websites (Zheng, Kohavi, & Mason, 2001). In these real datasets, the transaction size follows exponential distribution. Table 5 shows the characteristics of the real data, where T_m and T_a indicate the maximum size and average size of transactions, respectively.

Three sets of experiments are conducted. The first set of experiments is the basic experiments. We directly apply the method as used in synthetic data experiments. The parameters k (number of hash functions), m (length of Bloom filters), and n (average size of transactions, as shown in Table 5) are determined by $k = \frac{m}{n} \ln 2$. Our first set of experiments shows a decreasing trend of error rates with the length of Bloom filters. However, the error rates cannot be below 12% even with very long Bloom filters (960 bits).

In the second set of experiments, we improve our approach by introducing a new concept called *virtual* average size of transactions. The results show that we can achieve very low error rates (below 2%) with 640-bit Bloom filters.

In the third set of experiments, we propose a δ-folding technique to further reduce storage requirement. We can save at most 75% storage while keeping the error rates below 2%. Moreover, we save over half of running time in the best case in which the storage saving rates are close to the summit.

We use minimum support threshold $= 0.75\%$ and $\alpha = 0$ for all *three* sets of experiments.

5.2.1 Basic experiment

In this set of experiments, we directly apply our approach as used in synthetic data experiments. We vary the length of Bloom filters from 480 bits to 960 bits. We measure the error rate ε, which is defined as follows:

$$\varepsilon = \frac{\sum_{\ell=1} (|E_p| + |E_n|) \text{ of freqent } \ell\text{-itemsets by BF method}}{\sum_{\ell=1} \text{number of frequent } \ell\text{-itemsets by Apriori}} \times 100\%.$$

Figure 3 shows the decreasing trend of error rates with the length of Bloom filters. The figure illustrates that even with 960 bits, the error rates are still greater than 12%.

The high error rates are due to a larger portion of transactions whose sizes are much more greater than the average size (see Fig. 4 which gives the distribution of transaction size for both synthetic and real datasets). For synthetic datasets, the

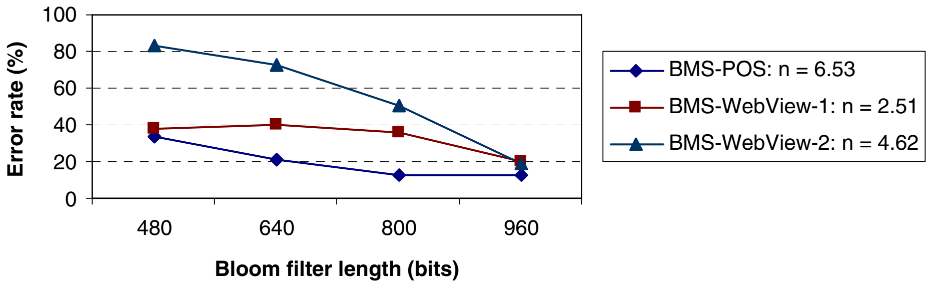


Fig. 3 Error rate w.r.t. Bloom filter length

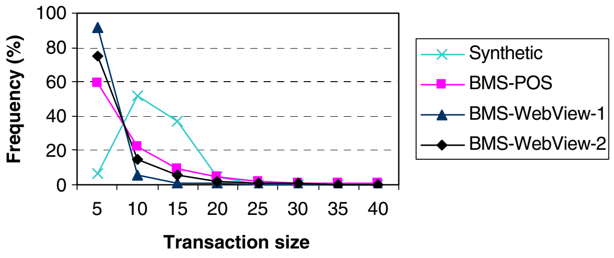
transaction size follows a Poisson distribution; over 99.8% of transactions have sizes between 1–20; less than 0.2% of transactions have sizes greater than 20 (see Table 1). For real datasets, the transaction size follows exponential distributions, relatively more transactions have sizes much greater than the average size (see Fig. 4 and Table 5). If such transactions are represented by Bloom filters, the number of 1s in a Bloom filter would greatly exceed the optimal value which is the half of the length of Bloom filters. This will increase the false positive rate in data mining from Bloom filters.

5.2.2 Improvement

To decrease the error rates, we introduce the concept of *virtual* average size of transactions. The virtual average size of transactions is the parameter n used in our data mining algorithm which may or may not be the "real" average size of transactions. Given virtual average size n, parameters k and m are still determined by $k = \frac{m}{n} \ln 2$.

Given a dataset, one can increase the virtual average size n so as to increase the mining precision. Figure 5 shows the error rates for different Bloom filter lengths (480–960 bits) and different virtual average size ($n = 10, 20, 30$). One can find that the error rates are no more than 2% for $n = 30$ and $m \geqslant 640$. This is a significant improvement compared with more than 12% error rates for up to 960 bits Bloom filters without using the virtual average size (see Fig. 3). In the next set of experiments, we generalize this method and try to save more storage space.

Fig. 4 Distribution of transaction sizes



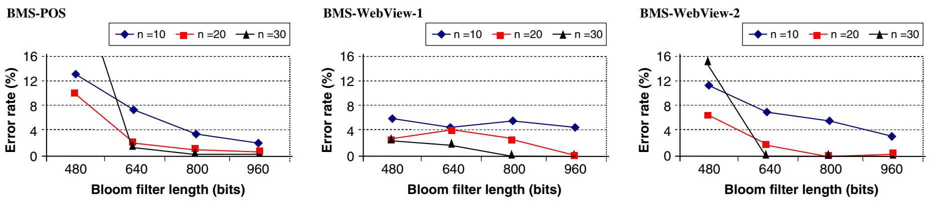


Fig. 5 Error rate w.r.t. Bloom filter length where $n = 10, 20, 30$

5.2.3 Storage saving

Looking into the distribution of transaction size (see Table 5), we find that few transactions have lengths greater than 30. This enlightens us to use shorter Bloom filters for small size transactions so as to save storage space. A simple way to achieve this is to categorize transactions into several clusters based on their sizes and generate Bloom filters separately for each cluster. However, this could increase communication costs between edge servers and clients. In our original solution, each time edge servers perform data mining on certain cluster, clients need to provide the servers with corresponding Bloom filters. To avoid transmitting different lengths of Bloom filters,only the longest Bloom filters of candidate itemsets are sent to edge servers at one time. Edges servers can convert these long Bloom filters to shorter ones when performing the mining process for different clusters.

In the following, we first provide a simple solution called δ-folding for converting Bloom filters, then we show how to use δ-folding in data mining process.

(1) *δ-folding* For convenience, let $bit(B, i)$ be the i^{th}-bit of Bloom filter B ($0 < i \leqslant m$), and $\vee$ be bitwise OR. Assume that δm is an integer where $0 < \delta \leqslant 1$.

Definition 5.1 [δ-folding] Given an m-bit Bloom filter B, δ-folding of B, denoted as B_δ, is defined as a δm-bit Bloom filter generated from B: for $0 < i \leqslant \delta m$, $bit(B_\delta, i) = bit(B, i) \bigvee_{\substack{0<j\leqslant m \\ j=i \bmod \delta m}} bit(B, j)$.

Consider a 10-bit Bloom filter B. $B_{0.7}$ is defined as $bit(B_{0.7}, i) = bit(B, i) \vee bit(B, i+7)$ for $i = 1, 2, 3$, and $bit(B_{0.7}, i) = bit(B, i)$ for $i = 4, 5, 6, 7$.

From the above definition, it is obvious that B_δ is a Bloom filter that is generated with the same hash functions that are used for generating B. Recall $k = \frac{m}{n} \ln 2$ (Lemma 2.3). With k unchanged, the *virtual* average size of transactions corresponding to B_δ is δn and the length of B_δ is δm.

(2) *Mining Process* Now consider a simple case in which data are partitioned into two groups. Note that it is straightforward to generalize it to multiple group case (e.g., using clustering methods). Group 1 contains transactions whose sizes are *greater* than δn and Group 2 contains other transactions, where δ is a parameter chosen to minimize the storage requirement. All transactions are firstly converted to Bloom filters of m bits long. Then δ-folding is applied to the Bloom filters of transactions in Group 2. As result, the outsourced database contains Bloom filters in two formats (m-bits and δm-bits). In mining process, clients send long Bloom filters (in m bits

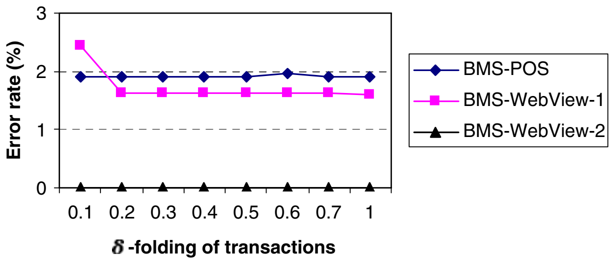


Fig. 6 Error rate of δ-folding where $m = 640$ and $n = 30$

format) of candidate itemsets to edge servers. When edge servers process transactions in δm-bit format, they apply δ-folding to all candidate Bloom filters so that the candidate Bloom filters have the same length as transaction Bloom filters.

With different formats of Bloom filters, the mining process is the same as our original algorithm except the counting phase. In the counting phase, candidate Bloom filters are checked first against transactions in a small group. Then the counting process moves to a large group where the frequencies obtained in the first group are accumulated and used as initial values.

(3) *Experimental Results* We have the following conclusions from our experiments in which δ-folding technique is applied: (a) no increasing of error rates, (b) significantly high rates of storage saving, (c) saving of over 50% running time in the best case, and requiring of nearly 70% extra running time in the worst case.

Figure 6 gives the experimental results for $m = 640$ and $n = 30$. For comparison, the figure also shows the error rates of 1-folding (i.e., no partition in data mining as shown in Fig. 5). It is clear that, with δ-folding, we achieve almost the same error rates (as low as 2%) for all *three* datasets as compared with the previous experiment.

More interestingly, we can save storage space significantly without degrading the mining precision. Figure 7 shows that the highest saving rates are reached at 0.2- or 0.3-folding. Saved storage space are over 75% of BMS-WebView-1 data, nearly 65% of BMS-WebView-2 data at 0.2-folding, and over 55% of BMS-POS data at 0.3-folding. The storage saving rate is defined as

$$\frac{\text{storage requirement with 1-folding} - \text{storage requirement with } \delta\text{-folding}}{\text{storage requirement with 1-folding (no partition in data mining)}} \times 100\%.$$

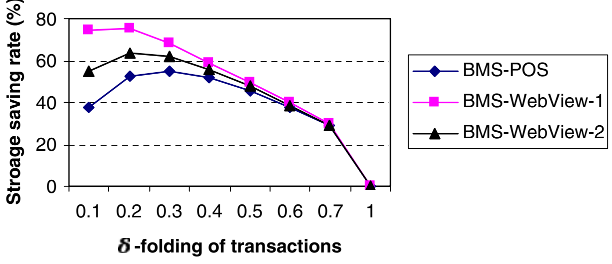


Fig. 7 Storage saving rate w.r.t. δ-folding

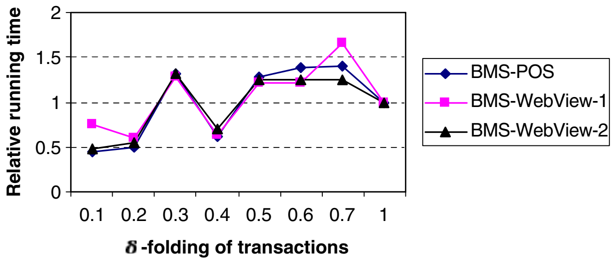


Fig. 8 Relative running time w.r.t. δ-folding

The change of saving rates are due to two factors. The first factor is the proportion of transactions which are represented by Bloom filters with δ-folding. The second factor is δ which determines the length of Bloom filters with δ-folding. For BMS-WebView-1 and BMS-WebView-2, the sizes of most transactions are less than six. The highest saving rate is achieved at 0.2-folding in which case data are partitioned at threshold six and most transactions are represented by short Bloom filters of 128 bits. For BMS-POS, the average transaction size is 6.53; there are still a lot of transactions whose sizes are greater than six. The highest saving rate is thus reached at 0.3-folding.

Let the running time of 1-folding be 1. Figure 8 shows the relative running time of δ-folding compared with 1-folding. The figure illustrates that δ-folding saves over half of running time in the best case and requires nearly 70% extra time in the worst case. When δ =0.1, 0.2 and 0.4, the running time is close to 0.5. In general, the running time of δ-folding is comparable to 1-folding (i.e., no partition of data in mining). One reason behind this is that after δ-folding, some transactions are represented by shorter Bloom filters. This could reduce the time of data fetching. On the other hand, with δ-folding, the hash-tree of candidates is generated twice in the counting phase (for different groups of transactions). This may require some extra time in data mining process.

6 Conclusions

We have proposed an approach for preserving privacy in association rule mining. The main idea of our method is to use keyed Bloom filters to represent transactions as well as data items so as to preserve privacy. We have also proposed a δ-folding technique to save storage requirement.

Our approach is different from previous solutions in that it can fully preserve data privacy while maintaining the precision of mining results. It can also be used to explore the tradeoff between mining precision and storage requirement. Rigorous experiments show that our approach can save storage requirement significantly without sacrificing much mining precision and running time.

In the future, we plan to study privacy preserving approaches to mining specific types of frequent itemsets, such as maximum itemsets and closed itemsets. It is also a parallel direction to study the robustness of Bloom filter method against attacks especially from a viewpoint of cryptography.

Appendix A: Proofs

Proof of Theorem 3.4. From Eq. 1, one can derive that the false positive rate for checking any single item $s \in S - T_i$ is p_1^k, where $p_1 = \left(1 - e^{-kn_i/m}\right)$ is the probability that a specific bit is 1 in $B(T_i)$ and k is the number of bits to which the item is hashed. From Lemma 3.3, we know that $S \cap T_i \subseteq_B T_i$ for sure. Since all items in $S - T_i$ are hashed to $||B(S - T_i)||$ bits, the false positive rate[4] f_i for checking S from $B(T_i)$ is $p_1^{||B(S-T_i)||} = \left(1 - e^{-kn_i/m}\right)^{||B(S-T_i)||}$. □

Proof of Theorem 3.7. Recall that $support(S) = Bsupport(S) - s_e$ and s_e is the sum of $N - support(S)$ 0–1 random variables with mean μ. If $Bsupport(S) \geqslant min_{sup} + \mu$, then $min\_sup = Bsupport(S) - \mu - \delta$ and thus $\Pr\left(support(S) \geqslant min\_sup\right) = \Pr(s_e \leqslant \mu + \delta) \geqslant 1 - \epsilon_1$ according to Lemma 3.6, where $\epsilon_1 = e^{\left(N - support(S)\right) \cdot H_{\bar{f}}\left(\bar{f} + \frac{\delta}{N - support(S)}\right)}$. Similarly, if $Bsupport(S) < min\_sup + \mu$, then $min\_sup = Bsupport(S) - \mu + \delta$ and thus $\Pr\left(support(S) < min\_sup\right) = \Pr(s_e > \mu - \delta) \geqslant 1 - \epsilon_2$, where $\epsilon_2 = e^{\left(N - support(S)\right) \cdot H_{1-\bar{f}}\left(1 - \bar{f} + \frac{\delta}{N - support(S)}\right)}$. □

Proof of Heuristic 4.1. All Bloom filters in a subtree share the same segment s as associated with the subtree's root. If any bit in s is 1 while the corresponding bit in $B(T_i)$ is 0, then $B(S) \wedge B(T_i) \neq B(S)$ for any Bloom filter $B(S)$ in the subtree. □

Appendix B: Estimate of false positive

Given Bloom filters $B(S)$ and $B(T_i)$, in what follows we consider how to compute false positive rate $f_i \approx \left(1 - e^{-kn_i/m}\right)^{||B(S-T_i)||}$. Note that $|S| \ll d$ and $|T_i| \ll d$ in data mining practice. If we model the composition of itemset S as random selection of $|S|$ items from $\mathcal{I} = \{I_1, \cdots, I_d\}$, then the probability that S and T_i share at least one item is

$$1 - \frac{d - |T_i|}{d} \cdot \frac{d - |T_i| - 1}{d - 1} \cdot \cdots \cdot \frac{d - |T_i| - |S| - 1}{d - |S| - 1} \approx 0$$

Therefore, we estimate

$$f_i \approx \left(1 - e^{-kn_i/m}\right)^{||B(S)||}$$

Now consider the average false positive rate

$$\bar{f} = \frac{1}{N - support(S)} \sum_{i=1}^{N - support(S)} f_i.$$

In most data mining practice, we have $support(S) \ll N$, and thus $\bar{f} \approx \frac{1}{N} \sum_{i=1}^{N} f_i$. From linear Taylor expansions, it is easy to obtain

$$\bar{f} \approx \left(1 - e^{-kn/m}\right)^{||B(S)||}$$

[4]One may consider to compute the false positive rate using $\left(1 - e^{-kn_i/m}\right)^{k|S-T_i|}$, which is the product of the false positive rate for checking each item in $S - T_i$. However, this method is *not* precisely correct because the length of Bloom filter is limited, and some of the items in $S - T_i$ may hash to same bits (i.e., $||B(S - T_i)|| \leqslant k|S - T_i|$); that is, checking each item of $S - T_i$ is not an independent event in computing the false positive rate.

where $n = \frac{1}{N}\sum_{i=1}^{N} n_i$ is the average transaction size. By choosing the optimal hash function number $k = \frac{m}{n}\ln 2$, one can further simplify the computation

$$\bar{f} \approx 0.5^{||B(S)||}$$

Discussion. The original intention of the revision of minimum support threshold is to increase the mining precision, especially to prevent some infrequent itemsets from appearing in the results. The above estimate is to give a rough extent of this kind of errors. Our experimental results (presented in Section 5) show that the revision of minimum support threshold slightly affects the mining precision. That is, if we do not apply the revision, there are only a few unexpected frequent itemsets appearing in the mining results as compared with the case where the revision is applied. On the other hand, with the revision some frequent itemsets may be missed out; however, this situation does not happen for the case without revision.

Acknowledgments Work of Wu was supported in part by USA National Science Foundation Grant CCR-0310974 and IIS-0546027. Thanks also go to the anonymous reviewers who provide helpful comments on earlier versions of this paper.

References

Agrawal, D., & Aggarwal, C. C. (2001). On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Santa Barbara, California (pp. 247–255).

Agrawal, R., Imilienski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Database* (pp. 207–216). New York: ACM Press.

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2004). Order preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Database*, Paris, France (pp. 563–574).

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *VLDB'94*, Santiago, Chile (pp. 487–499).

Agrawal, R., & Srikant, R. (2000). Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Dallas, Texas (pp. 439–450).

Atallah, M., Bertino, E., Elmagarmid, A. K., Ibrahim, M., & Verykios, V. S. (1999). Disclosure limitation of sensitive rules. In *Proceedings of the IEEE Knowledge and Data Engineering Exchange Workshop*, Chicago, Illinois (pp. 45–52).

Bloom, B. (1970) Space time tradeoffs in hash coding with allowable errors. *Communications of theACM*, *13*(7), 422–426.

Border, A. Z., & Mitzenmacher, M. (2002). Network applications of bloom filters: A survey. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*, Urbana-Champaign, Illinois (pp. 636–646).

Chernoff, H. (1952). A measure of asymptotic efficiency for tests based on the sum of observations. *Annals of Mathematical Statistics*, *23*, 493–509.

Cohen, S., & Matias, Y. (2003). Spectral bloom filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Database*, San Diego, California (pp. 241–252).

Dasseni, E., Verykios, V. S., Elmagarmid, A. K., & Bertino, E. (2001). Hiding association rules by using confidence and support. In *Proceedings of the 4th International Information Hiding Workshop*, Pittsburg, Pennsylvania (pp. 369–383).

Du, W., & Atallah, M. J. (2001). Secure multi-party computation problems and their applications: A review and open problems. In *Proceedings of New Security Paradigms Workshop 2001*, Cloudcroft, New Mexico (pp. 11–20).

Du, W., & Zhan, Z. (2002). Building decision tree classifier on private data. In *Proceedings of IEEE ICDM'02 Workshop on Privacy, Security, and Data Mining*, volume 14, Maebashi City, Japan (pp. 1–8).

Evfimievski, A.,Srikant, R., Agrawal, R., & Gehrke, J. (2002). Privacy preserving mining of association rules. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada (pp. 217–228).

Evfimievski, A., Gehrke, J., & Srikant, R. (2003). Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database System*, San Diego, California (pp. 211–222).

Fan, L., Cao, P., Almeida, J., & Border, A. Z. (2000). Summary cache: A scalable wide-area web cachesharing protocol. *IEEE/ACM Transactions on Networking*, *8*(3), 281–293.

Hacigumus, H., Iyer, B., Li, C., & Mehrotra, S. (2002a). Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD International Conference on Management of Database*, Madison, Wisconsin (pp. 216–227).

Hacigumus, H., Iyer, B., & Mehrotra, S. (2002b). Providing database as a service. In *Proceedings of the International Conference on Data Engineering*, San Jose, California (pp. 29–40).

Hacigumus, H., Iyer, B., & Mehrotra, S. (2004). Efficient execution of aggregation queries over encrypted relational databases. In *Proceedings of International Conference on Database Systems for Advanced Applications*, (pp. 125–136). Jeju Island, Korea.

Hoeffding, W. (1963). Probability for sums of bounded random variables. *Journal of the American Statistical Association*, *58*, 13–30.

Iyer, B., Mehrotra, S., Mykletun, E., Tsudik, G., & Wu, Y. (2004). A framework for efficient storagesecurity in RDBMS. In *Proceedings of International Conference on EDBT*, Crete, Greece (pp. 147–164).

Kantarcıoğlu, M., & Clifton, C. (2002). Privacy preserving distributed mining of association rules on horizontally partitioned data. In *Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Madison, Wisconsin (pp. 24–31).

Kantarcıoğlu, M., Jin, J., & Clifton, C. (2004). When do data mining results violate privacy? In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, Washington (pp. 599–604).

Kargupta, H., Datta, S., Wang, Q., & Sivakumar, K. (2003). On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the 3rd International Conference on Data Mining*, Melbourne, Florida (pp. 99–106).

Li, Z., & Ross, K. A. (1995). PERF join: An alternative to semijoin and Bloom join. In *Proceedings of the International Conference on Information and Knowledge Management*, Baltimore, Maryland (pp. 137–144).

Lindell, Y., & Pinkas, B. (2002). Privacy preserving data mining. *Journal of Cryptology*, *15*(3), 177–206.

Mullin, J. K. (1990). Optimal semijoins for distributed database systems. *IEEE Transactions on Software Engineering*, *16*(5), 558–560.

Mykletun, E., Narasimha, M., & Tsudik, G. (2004). Authentication and integrity in outsourced databases. In *Proceedings of the 11th ISOC Annual Network and Distributed System Security Symposium*, San Diego, California.

Oliveira, S., & Zaiane, O. (2002). Privacy preserving frequent itemset mining. In *Proceedings of the IEEE ICDM Workshop on Privacy, Security and Data Mining*, Maebashi City, Japan (pp. 43–54).

Oliveira, S., & Zaiane, O. (2003a). Algorithms for balancing privacy and knowledge discovery in association rule mining. In *Proceedings of the 7th International Database Engineering and Applications Symposium*, Hongkong, China (pp. 54–63).

Oliveira, S., & Zaiane, O. (2003b). Protecting sensitive knowledge by data sanitization. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, Melbourne, Florida (pp. 211–218).

Pang, H., & Tan, K. L. (2004). Authenticating query results in edge computing. In *Proceedings ofthe 20th International Conference on Data Engineering*, Boston, Massachusetts (pp. 560–571).

Pinkas, B. (2002). Cryptographic techniques for privacy preserving data mining. *ACM SIGKDDExplorations*, *4*(2), 12–19.

Rizvi, S., & Haritsa, J. (2002). Maintaining data privacy in association rule mining. In *VLDB'02*, Hongkong, China (pp. 682–693).

Saygin, Y., Verykios, V. S., & Clifton, C. (2001). Using unknowns to prevent discovery of association rules. *Sigmod Record*, *30*(4), 45–54.

Vaidya, J., & Clifton, C. (2002). Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Alberta, Canada (pp. 639–644).

Yao, A. (1986). How to generate and exchange secrets. In *Proceedings of the 27th IEEEFOCS*, Ontario, Canada (pp. 162–167).

Zheng, Z., Kohavi, R., & Mason, L. (2001).Real world performance of association rule algorithms.In *Proceedings of the 7th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California (pp. 401–406).