Finding Combinations of Binary Variables with Guaranteed Accuracy

Yoshito Baba[1] **Mahito Sugiyama**[1,2] **Takashi Washio**[1]
[1]The Institute of Scientific and Industrial Research, Osaka University, Japan
[2]JST, PRESTO, Japan
{baba,mahito,washio}@ar.sanken.osaka-u.ac.jp

Abstract

We propose a probabilistic approach to finding frequently occurring combinations of binary variables with accuracy guarantees, which is a computationally challenging task due to the combinatorial explosion of variable combinations. For recently appearing massive databases over several terabytes, existing methods are too exhaustive to enumerate frequent combinations as they cannot even load such databases on the main memory. In contrast to such exhaustive approaches, our method efficiently finds frequent variable combinations via iterative sampling and does not need to load the entire database. Moreover, we can theoretically control the ratio of false positives and false negatives.

1 Introduction

Finding interactions or associations between binary variables is an important problem in a wide range of applications. Examples include finding frequently co-purchased items in market basket analysis [2, 3], detecting combinations of single nucleotide polymorphisms (SNPs) in genome-wide association studies (GWAS) [4, 8], and discovering neuron combinations from neural firing patterns [6]. To date, this task has been intensively studied as *frequent itemset mining* in the data mining field [1], where a variable combination is called an *itemset*. However, to analyze recently appearing massive databases, which can be over several terabytes, existing methods are too exhaustive because they cannot even load such databases on the main memory.

Here we propose an efficient sampling-based approach for finding frequently co-occurring combinations of binary variables from massive databases. Our algorithm finds variable combinations from a randomly and independently sampled subset of a given dataset without seeing the entire dataset, resulting in the complexity which is independent of the dataset size. Moreover, we control both the false positive and false negative rates: the false negative rate is controlled by iterative sampling and the false positive rate is controlled by integrating batches of candidate combinations. The number of iteration of sampling and the number of batches are directly computed from the desired accuracy.

This paper is organized as follows: Followed by the review of related work in Section 2, we formulate our problem of finding frequent combinations of binary variables in Section 3, and present the proposed algorithm in Section 4 with theoretical analysis. We empirically examine the performance of our method in Section 5 and conclude the paper in Section 6.

2 Related Work

The Apriori algorithm [2] is the first method which enables us to efficiently find frequent itemsets by focusing on the property that any super set of an infrequent itemset cannot be frequent. This property is fundamental in a number of recently proposed methods including [3]. One of the fastest

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

methods for frequent itemset mining is LCM [9], which uses advanced data structures such as arrays, bitmaps, and prefix trees, and won the competition "FIMI04". In this paper, we compare our method with LCM to verify the efficiency of our algorithm.

Exhaustive approaches including Apriori and LCM need to store the whole database $\mathcal{D}$ in the main memory, hence if $\mathcal{D}$ is too massive to store in memory, efficient mining is difficult. Although sequential processing that incrementally reads transactions $\mathcal{D}$ one after another (e.g. [5]) can solve the above problem, there is no efficient incremental method that can treat massive databases.

In contrast to exhaustive approaches that always find all frequent itemsets, approximation methods have been studied for more efficient mining. Random Intersection Trees (RIT) [7] is one of the representative methods, which approximately finds frequent itemsets from intersections of transactions randomly sampled from a database. RIT constructs trees from random subsamples, and the leaf nodes correspond to frequent itemsets. In this paper, we use an idea of subsampling from this technique and exclude tree construction to directly generate frequent itemsets without unnecessary computation.

3 Problem Formulation

Let $\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be a dataset such that each sample $\boldsymbol{x}_i = (x_i^1, x_i^2, \ldots, x_i^n) \in \{0,1\}^n$ is an n-dimensional binary vector. For the set of feature indices $A = \{1, 2, \ldots, n\}$, we define the combination for each subset $F \subseteq A$ as $\prod_{j \in F} x_i^j \in \{0,1\}$ for $i \in \{1, 2, \ldots, N\}$. Then the *frequency* $f(F)$ of a combination F is given as

$$f(F) = \frac{1}{N} \sum_{i=1}^{N} \prod_{j \in F} x_i^j. \tag{1}$$

In the data mining terminology, each $j \in A$ and $F \subseteq A$ are called an *item* and an *itemset*, respectively, and $T(\boldsymbol{x}_i) = \{j \in A \mid x_i^j = 1\}$, which is the set of indices for "1" of $\boldsymbol{x}_i$, is called a *transaction*. The frequency $f(F)$ of F is often called the *support* of F.

Our objective is to find frequently occurring combinations, that is, find the set $\{F \subseteq A \mid f(F) \geq minsup\}$, where $minsup$ is given by the user. Each frequent combination F is called a *frequent itemset*. In particular, we focus on finding a *maximal* frequent itemset, which is a frequent itemset F such that $f(F) \geq minsup$ while $f(F \cup \{j\}) < minsup$ for any $j \in A \setminus F$.

4 The Proposed Algorithm

We propose a sampling-based algorithm to find frequent itemsets (combinations of features), whose time and space complexities are independent of the size $|\mathcal{D}|$ of a dataset. By using two thresholds, $minsup$ and $lowsup$ with $lowsup \leq minsup$, our algorithm can control the probability of finding itemsets with the frequency $minsup$ while not finding itemsets whose frequency is $lowsup$. Such probabilities are controlled by the number r of iteration of sampling and the number k of batches. We introduce our algorithm using these two parameters in Section 4.1, followed by showing the link between these parameters and two thresholds in Section 4.2.

4.1 Algorithm Overview

First we randomly and independently sample r vectors $\boldsymbol{x}_{i_1} \boldsymbol{x}_{i_2} \ldots \boldsymbol{x}_{i_r}$ from $\mathcal{D}$ and obtain the intersection vector $\boldsymbol{s}_i$ such that $T(\boldsymbol{s}_i) = T(\boldsymbol{x}_{i_1}) \cap T(\boldsymbol{x}_{i_2}) \cap \cdots \cap T(\boldsymbol{x}_{i_r})$. We repeat this process and get h intersection vectors $\mathcal{S} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_h\}$. Then we make a batch $\mathcal{B} \subseteq \mathcal{S}$ consisting of vectors that are the maximal subset of $\mathcal{S}$ with respect to the feature set $T(\boldsymbol{s})$, that is, for each $\boldsymbol{s}_i \in \mathcal{S}$, we remove it from $\mathcal{S}$ if $T(\boldsymbol{s}_i) \subseteq T(\boldsymbol{s}_j)$ for some $\boldsymbol{s}_j \in \mathcal{S}$.

By repeating the above operation for k times, we obtain k batches $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$, and construct the final output $\mathcal{F}$ by integrating the k batches one by one. Let $\mathcal{F} = \mathcal{B}_1$. Whenever we obtain a new batch $\mathcal{B}_i$, we integrate the current $\mathcal{F}$ and the new $\mathcal{B}_i$ by the following three steps: (1) Start from $\mathcal{F}_{\text{new}} = \emptyset$. For every pair $(\boldsymbol{s}, \boldsymbol{t}) \in \mathcal{F} \times \mathcal{B}_i$, put $\boldsymbol{u}$ to $\mathcal{F}_{\text{new}}$ such that $T(\boldsymbol{u}) = T(\boldsymbol{s}) \cap T(\boldsymbol{t})$. (2) After checking all pairs in $\mathcal{F} \times \mathcal{B}_i$, for each $\boldsymbol{u} \in \mathcal{F}_{\text{new}}$, we remove it if $T(\boldsymbol{u}) \subseteq T(\boldsymbol{u}')$ for some $\boldsymbol{u}' \in \mathcal{F}_{\text{new}}$. (3) Set $\mathcal{F} = \mathcal{F}_{\text{new}}$.

The time complexity of computing an intersection vector is $O(rn)$, and that of computing a batch and batch integration is $O(nh^2)$. Therefore, the overall time complexity is $O(rnh + nh^k)$ in the worst case. The worst space complexity is $O(nh^k)$ as we integrate batches one by one[1]. Thus both time and space complexities are independent of $|\mathcal{D}|$.

4.2 Theoretical Analysis

We present a probabilistic analysis to examine the accuracy of our algorithm in finding frequent itemsets (variable combinations) and show that two parameters of the algorithm, r and k, can be determined from the desired accuracy.

Given a frequency σ with assuming that σn is a natural number. For an itemset $I \subseteq A$ with $f(I) = \sigma$, the probability of having $I \subseteq T(\boldsymbol{s})$ for an intersection vector $\boldsymbol{s}$ is given as

$$p_S(\sigma, r) = \binom{\sigma n}{r} \Big/ \binom{n}{r}.$$

For h intersection vectors $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_h$, the probability that at least one vector $\boldsymbol{s}_i$ satisfying $I \subseteq T(\boldsymbol{s}_i)$ exists is obtained as $p_R(\sigma, r, h) = 1 - (1 - p_S(\sigma, r))^h$. Thus, by setting $\varepsilon = 1 - p_R(\sigma, r, h)$, called the *error ratio*, we can compute h from ε and $minsup$:

$$h = \frac{\log \varepsilon}{\log(1 - p_S(minsup, r))}.$$

Moreover, by generating k batches, the probability of I being included in all k batches is given as

$$p_B(\sigma, r, h, k) = p_R(\sigma, r, h)^k = \Big(1 - (1 - p_S(\sigma, r))^h\Big)^k.$$

Therefore, our algorithm can find itemsets with the frequency $minsup$ with controlling the false negative rate as $1 - p_B(minsup, r, h, k)$, while keeping the false positive rate of itemsets with $lowsup$ as $p_B(lowsup, r, h, k)$. To automatically determine the number k of batches, we use

$$\delta = p_B(minsup, r, h, k) - p_B(lowsup, r, h, k) \tag{2}$$

and set as $k_{\mathrm{opt}} = \mathrm{argmax}_k\, \delta$. Finally, we have two parameters r and ε and two thresholds $minsup$ and $lowsup$, and h, k_{opt}, and two probabilities $p_B(minsup, r, h, k)$ and $p_B(lowsup, r, h, k)$ are determined from them. Notice that we can also obtain ε from the desired $p_B(minsup, r, h, k)$.

5 Experiments

Environment. We performed our algorithm on Windows10 Pro 64bit OS with the processor Intel Core i7-6500U CPU 2.50GHz and the 16GB main memory. The proposed algorithm is implemented in C/C++, and we compared it with LCM[2] implemented in C. All codes were compiled in gcc 4.9.3.

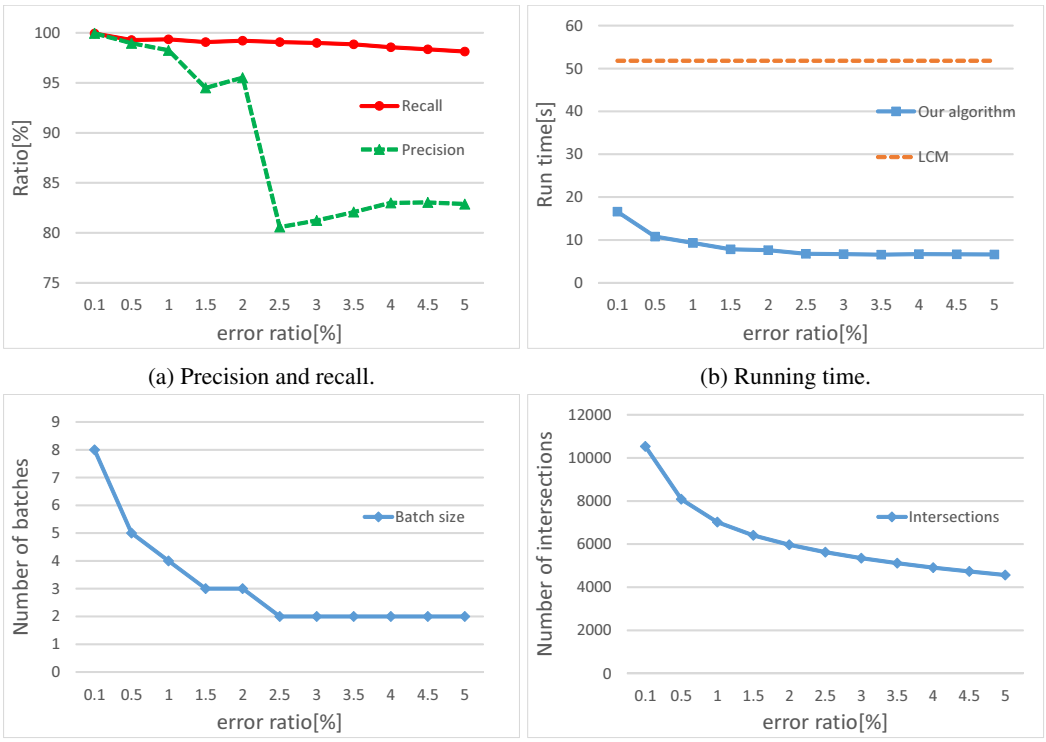
Protocol. We generated synthetic data for evaluation. In the dataset $\mathcal{D}$, $|\mathcal{D}| = 1,000,000$ and $n = 100$, and each binary vector $\boldsymbol{x} \in \mathcal{D}$ was randomly generated with $|T(\boldsymbol{x})| = 40$, which means that this dataset is relatively dense. Using this dataset, we evaluated running time, recall, and precision with respect to changes in ε that controls the false positive and negative rates with fixing the samples size $r = 8$ and two thresholds $minsup = 0.4$ and $lowsup = 0.3$. We adopted LCM ver. 5.3 [9] as a comparison partner as it is known to be the fastest exhaustive frequent itemset mining algorithm. Throughout our experiment, we removed features $j \in A$ whose frequencies $f(\{j\})$ are smaller than $minsup$ as preprocessing as with LCM for the fair comparison. We also examined RIT [7], but it was significantly slower than both our algorithm and LCM, hence we skip results of RIT.

Evaluation Criteria. We use precision and recall with respect to $minsup$ and $lowsup$ to evaluate the effectivenss of our algorithm. More specifically, let $\mathcal{F}_{minsup}$ be the set of frequent itemsets whose frequencies are higher than $minsup$. Since our method finds maximal itemsets, precision and recall are defined as:

$$\text{Precision} = |\{\boldsymbol{x} \in \mathcal{F} \mid f(\boldsymbol{x}) \geq lowsup\}| \,/\, |\mathcal{F}|,$$
$$\text{Recall} = |\{\boldsymbol{x} \in \mathcal{F}_{minsup} \mid T(\boldsymbol{x}) \subseteq T(\boldsymbol{y}) \text{ for some } \boldsymbol{y} \in \mathcal{F}\}| \,/\, |\mathcal{F}_{minsup}|.$$

[1] In practice, the time and space complexities become almost $O(rnh + nh^2)$ and $O(nh)$, respectively, due to the similarity of each $\mathcal{B}_i$ and the maximality of $\mathcal{F}_{\mathrm{new}}$.

[2] `http://research.nii.ac.jp/~uno/code/lcm53.zip`



(a) Precision and recall.

(b) Running time.

(c) The number of batches k.

(d) The number of intersections h.

Figure 1: Experimental results (a), (b) and automatically determined parameters (c), (d).

Note that precision is obtained from $lowsup$ because our method is designed not to find itemsets whose frequencies are smaller than $lowsup$.

Results. We plot precision and recall in Figure 1a and running time in Figure 1b obtained from our synthetic data. We also show the number of batches and the number of intersections in Figures 1c and 1d respectively, which are theoretically computed. Figure 1a shows that our algorithm has high precision and recall when the error ratio ε is small. These results are consistent with our theoretical analysis. In particular, if the error ratio $\varepsilon = 1.0$, both precision and recall are kept with values higher than 0.97, while our algorithm is more than five times faster than LCM as shown in Figure 1b. In addition, we can confirm that precision increases if the number of batches increases in Figure 1c. Note that a non-monotonic behavior of the precision curve is caused by the discrete change of the batch size.

Discussion. Our results show that the proposed algorithm can find frequent itemsets faster than LCM with the low tolerance ε of the error which is used to determine the number k of batches and the number h of intersections. Since the time and the space complexities of our algorithm is independent of the size $|\mathcal{D}|$ of a dataset, our algorithm is expected to be much efficient for more massive datasets. Moreover, we confirm that our algorithm can control recall and precision with the same level by theoretical analysis without computing the actual frequencies of itemsets.

6 Conclusion

In this paper, we have proposed a new algorithm to efficiently find frequently occurring variable combinations (itemsets) using random sampling. We have theoretically shown the probability of finding frequent itemsets without computing the actual frequencies, and empirically shown that that the proposed algorithm is faster than the fastest exhaustive algorithm and can keep low false positive and negative rates. Our future work is to evaluate our method on real-world massive datasets which is difficult to store in main memory. Another future work is to improve the batch integration algorithm for more efficient computation.

References

[1] Agrawal, C. C. and Han, J., Eds. *Frequent Pattern Mining*. Springer, 2014.

[2] Agrawal, R. and Slikant, R. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, 487–499, 1994.

[3] Han, J., Pei, J., and Yin, Y. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 1–12, 2000.

[4] Llinares-López, F., Grimm, D. G., Bodenham, D. A., Gieraths, U., Sugiyama, M., Rowan, B., and Borgwardt, K. M. Genome-wide detection of intervals of genetic heterogeneity associated with complex traits. *Bioinformatics*, 31(12):i240–i249, 2015.

[5] Otey, M. E., Parthasarathy, S., Wang, C., Veloso, A., and Meira, W. Parallel and distributed methods for incremental frequent itemset mining. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(6):2439–2450, 2004.

[6] Picado-Muiño, D., Borgelt, C., Berger, D., Gerstein, G., and Grün, S. Finding neural assemblies with frequent item set mining. *Frontiers in neuroinformatics*, 7(9), 2013.

[7] Shah, R. D. and Meinshausen, N. Random intersection trees. *Journal of Machine Learning Research, 15, 629-654*, 629–654, 2014.

[8] Terada, A., Okada-Hatakeyama, M., Tsuda, K., and Sese, J. Statistical significance of combinatorial regulations. *PNAS*, 110(32):12996–13001, 2013.

[9] Uno, T., Asai, T., Uchida, Y., and Arimura, H. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, volume 3245 of *Lecture Notes in Computer Science*, 16–31, 2004.