Support Estimation in Frequent Itemset Mining by Locality Sensitive Hashing

Annika Pick[1,3,⋆], Tamás Horváth[2,1,3], and Stefan Wrobel[1,2,3]

[1] Fraunhofer IAIS, Sankt Augustin, Germany
[2] Dep. of Computer Science, University of Bonn, Germany
[3] Fraunhofer Center for Machine Learning, Sankt Augustin, Germany

Abstract. The main computational effort in generating all frequent itemsets in a transactional database is in the step of deciding whether an itemset is frequent, or not. We present a method for estimating itemset supports with two-sided error. In a preprocessing step our algorithm first partitions the database into groups of similar transactions by using locality sensitive hashing and calculates a summary for each of these groups. The support of a query itemset is then estimated by means of these summaries. Our preliminary empirical results indicate that the proposed method results in a speed-up of up to a factor of 50 on large datasets. The F-measure of the output patterns varies between 0.83 and 0.99.

Keywords: frequent itemset mining · locality sensitive hashing

1 Introduction

Frequent itemset mining, the first step of listing all association rules (see, e.g., [2, 9]) is one of the historical core problems of data mining and knowledge discovery. The most time-consuming step of all frequent itemset mining algorithms is to decide the frequency of itemsets, i.e., whether an itemset is supported by at least a certain number of transactions in the input database, or not. In the case that the mining algorithm has access to the transaction database only via queries of the form *"Is itemset X frequent?"*, all deterministic algorithms solving the frequent itemset mining problem correctly are required to check *all* itemsets in the *border* of frequent itemsets for frequency [9].

Checking if an itemset is frequent is often treated as a query to an *oracle* (or black box), focusing only on the answer rather than on its computational aspects. In contrast, we concentrate on the algorithmic issues of deciding whether an itemset is frequent, or not. In particular, in this *work-in-progress* paper we propose an algorithm that decides itemset frequency in a runtime that is guaranteed to be faster by at least a constant factor than the brute-force linear time algorithm solving this problem. Although our algorithm remains linear in the

[⋆] annika.pick@iais.fraunhofer.de

combined size of the input database and the query itemset, the speed-up (up to a factor of 50) obtained on large benchmark datasets is a remarkable improvement.

To obtain this improvement, we give up correctness and only require the (binary valued) answer returned to be correct with high probability. We achieve this goal by utilizing the advantageous algorithmic properties of *locality sensitive hashing* (LSH) [7]. Its main feature is that for a given metric space, elements with a small distance have a high probability of being hashed to the same bucket, while elements with a large distance have a low probability of collision. In the algorithm proposed in this work, transactions are grouped by using LSH based on the Hamming distance; we utilize the basic fact that the characteristic vectors of the subsets of a universe of cardinality d can be regarded as the vertices of the d-dimensional Hamming cube. Using the standard probability amplification techniques for LSH [7], in a preprocessing step we devise a data structure partitioning the input transaction database into blocks. With high probability, transactions with a small Hamming distance will be hashed to the same block, while those with a large Hamming distance to different ones. We remove all blocks of size less than c, where c is some user specified parameter, and calculate a weighted *summary* itemset for each remaining block. In this way, we compress the transaction database by a factor of at least c. For a query itemset X we then decide its frequency by matching X with all such summaries. Our algorithm estimates itemset support with two-sided error in this way. The error probability can, however, be controlled by standard amplification techniques [7].

Our *preliminary* empirical results indicate that this technique outperforms the fastest *exact* method in runtime on four out of six benchmark datasets, reaching a speed-up factor of up to 50. It is slower only on two (out of six) datasets; the slow-down is, however, marginal. In the long version we experimentally demonstrate that our method is especially suitable for datasets that are large and have a relatively small Hamming distance on average; these properties can quickly be checked (the latter e.g. by sampling) before running the algorithm. Regarding the estimation quality of our algorithm, we obtained an F-score of at least 80% on all datasets. Our algorithm is governed by different parameters; we also propose some simple heuristics for the choice of their values.

Finally we note that LSH has already been applied to frequent pattern discovery, however, in entirely different contexts. In particular, it was used in reducing the computational effort in the candidate generation step of the Apriori algorithm [3]. This application uses LSH to examine only pairs of elements that are likely to be similar.

2 Support Estimation using LSH

Our LSH-based algorithm for rapid frequency query evaluation consists of two main parts: In a *preprocessing* step we first compress the input database by partitioning it into blocks containing similar transactions and compute a summary for each of these blocks. In the *query answering* step, we estimate the support of the itemsets at hand by means of these summaries.

The Preprocessing Step. The preprocessing routine takes as input a transactional database $\mathcal{D}$ over $I = \{1, \ldots, m\}$ and filters out infrequent items so they do not influence the hashing procedure. It then builds L data structures, corresponding to the OR-amplification in LSH [7]. For each such data structure, we first select k functions $h_{i_1}, \ldots, h_{i_k}$ independently and uniformly at random, where h_i returns the ith entry $(1 \leq i \leq m)$, and assemble them into a function g as follows: g hashes two transactions $X, Y \subseteq I$ into the same bucket iff $h_{i_j}(X) = h_{i_j}(Y)$ (i.e., either $i_j \in X \cap Y$ or $i_j \notin X \cup Y$) for all $j = 1, \ldots, k$. Thus, we enforce two transactions in the same bucket to agree on all of the items in $\{i_1, \ldots, i_k\}$. Notice that two transactions in the same bucket can agree on even more items if the items are dependent, i. e., occur frequently together; this is precisely the case we are interested in. This construction of g corresponds to the AND-amplification in LSH [7].

In our empirical evaluation we found that there were many singleton buckets, which clearly do not represent any interesting pattern. In general, we discard all buckets with less than c elements, where c is an input parameter. Its choice is crucial for the speed-up. For each remaining bucket, we only store one representative transaction. Our heuristic to define it is to include only those items that are contained in a high fraction of the transactions in the bucket. This is controlled by parameter d: An item $i \in \{1, \ldots, w\}$ is included in the representative transaction of bucket b if at least a fraction d of the transactions hashed into b contain i. A heuristic for the choice of parameters c and d is proposed in the next section.

Since storing only the representative transactions and ignoring the bucket size is insufficient, we need to weight the summaries. A straightforward choice would be just using the bucket size, but we empirically validated that it is better to use the lower bound for how often the items in the representative transaction must have occurred together. For space limitations we omit these details. When this bound is positive, we use it as a weight and add the weighted representative transaction to the output database. For further speed-up, the output databases are stored in vertical form in a way similar to [11].

Answering Frequency Queries. Using the compressed databases constructed above, we evaluate frequency queries as follows. We compensate the information loss in the compressed databases by using a smaller frequency threshold $\theta_{\mathrm{r}} = \theta\epsilon$ for some $\epsilon \in (0, 1]$. An itemset is then regarded frequent in the original dataset w.r.t. θ if it is frequent in at least one of the L databases w.r.t. θ_{r}.

3 Empirical Evaluation

For the experimental evaluation, we compared the runtime of our method with that of Eclat [11], which was the fastest algorithm we considered for reference.

			Preproc. (s)		Query (µs)		Query	LSH Scores (%)		
Dataset	Transact.	Thr. θ	Eclat	LSH	Eclat	LSH	Speed-up	Prec.	Rec.	F
`AR-tool100k`	1,000,000	1%	0.17	38	981	432	2.3	94	74	**83**
`AR-tool1M`	990,000	1%	1.81	380	13915	279	49.9	92	82	**87**
`Kosarak`	100,000	1%	2.17	141	1520	37	44.5	87	80	**83**
`Retail`	100,000	0.1%	0.22	33	24	30	-1.3	92	86	**89**
`T10I4D100K`	88,162	0.6%	0.23	284	27	29	-1.1	97	86	**91**
`Mushroom`	8124	20%	0.03	3	468	17	27.2	99	100	**99**

Table 1. Performance of Eclat [11] vs. our algorithm on the six datasets. Number of transactions and frequency threshold θ in columns 2–3. Runtime of preprocessing and query evaluation are reported in columns 4–7; the respective speed-up of query evaluation in column 8. Rightmost three columns: precision, recall and F-score (all in %) obtained by our method; all are 100% for Eclat.

Regarding the *datasets*, we used four benchmark datasets [1, 5] to compare our algorithm with other approaches, as well as two newly generated synthetic datasets to examine the impact of database length on runtime. For the latter, we used the synthetic data generator contained in the ARtool project [4] and, except for the length, a fixed set of parameters. We call the resulting sets `AR-tool100k` (100,000 transactions) and `AR-tool1M` (one million transactions). The frequency parameter was set in the same magnitude as in [6, 8, 10], except for `Mushroom`. For this dataset we had to double the threshold used in [10], as the original threshold leads to an enormous output set.

Our algorithm depends on different *parameters*. For their choice, we randomly selected one of the datasets (`AR-tool100k`) and optimized the parameter choice for minimal information loss (parameter k) and maximal speed-up while maintaining an F-score above 80 %. As a result, we suggest the following heuristics to choose parameters: For parameters k and c, take a small sample of the new dataset and evaluate the number of deleted item occurrences for different values. As starting points, we recommend setting k to half of the number of frequent single items and c to 0.01% of the number of transactions in the dataset. Setting $d = 0.9$ and $L = 25$, for ϵ take the relative amount of item occurrences which have *not* been deleted. This corresponds to around $\epsilon = 0.35$ on `AR-tool100k`. Though even these simple heuristics lead to remarkable experimental results, in the long version of this paper we are going to develop some theoretically well-founded, more sophisticated strategies for the parameter choice.

For the *experimental evaluation*, we considered only itemsets with more than one element that were either frequent or minimal infrequent, as only these types of itemsets are checked for frequency by most frequent itemset mining algorithms. As mentioned above, we used the support counting method of Eclat [11] as a reference. Our first results with the six datasets are summarized in Table 1. Our approach is significantly faster on three datasets and is comparably fast on the other ones, all with an F-score of above 80 %.

4 Remarks

There are three properties influencing the performance of our algorithm. Firstly, (very) large datasets benefit more from our approach; this can be observed on the two ARtool datasets, which were generated with the same parameters except for the length. The dataset with one million transactions reaches a speed-up factor of nearly 50 vs. only factor 2.3 for the one with 100,000 transactions. Secondly, the average distance between two transactions in the database is also crucial because merging transactions in a bucket works better with more similar transactions. Thirdly, itemsets returned as infrequent slow down the algorithm, as all L compressed databases have to be checked. Overall, checking the three properties enables us to assess if a dataset is suitable for our algorithm.

Our algorithm can further be accelerated. In particular, for infrequent itemsets, all L compressed databases must be checked. Instead, we could break earlier, e.g., when a query itemset is not contained at all in the first few databases or its frequency is far below the threshold in them. Regarding the quality of the output, it would be interesting to understand the combinatorial properties of the transactions within a block that result in more accurate weighted summaries. Another challenge is to develop some theoretically well-founded method able to find the optimal parameter values for the dataset at hand in practically feasible time. Finally, for datasets over large sets of items it would be interesting to modify our algorithm to min-hashing by considering Jaccard similarity.

References

1. Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data/
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB Conference. vol. 1215, pp. 487–499 (1994)
3. Bera, D., Pratap, R.: Frequent-itemset mining using locality-sensitive hashing. In: International Computing and Combinatorics Conf. pp. 143–155. Springer (2016)
4. Cristofor, L.: The ARtool project. https://www.cs.umb.edu/ laur/ARtool/
5. De Bie, T.: An information theoretic framework for data mining. In: Proc. of the 17th ACM Int. Conf. on Knowledge Discovery and Data Mining. pp. 564–572. ACM (2011)
6. Goethals, B.: Survey on frequent pattern mining. Univ. of Helsinki **19**, 840–852 (2003)
7. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613. ACM (1998)
8. Lin, J.C.W., Hong, T.P., Gan, W., Chen, H.Y., Li, S.T.: Incrementally updating the discovered sequential patterns based on pre-large concept. Intelligent Data Analysis **19**(5), 1071–1089 (2015)
9. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data mining and knowledge discovery **1**(3), 241–258 (1997)
10. Moens, S., Aksehirli, E., Goethals, B.: Frequent itemset mining for big data. In: BigData Conference. pp. 111–118 (2013)
11. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: Parallel algorithms for discovery of association rules. Data mining and knowledge discovery **1**(4), 343–373 (1997)