An Adaptive Algorithm for Finding Frequent Sets in Landmark Windows

Xuan Hong Dang[1], Kok-Leong Ong[2], and Vincent Lee[3]

[1] Dept. of Computer Science, Aarhus University, Denmark
dang@cs.au.dk
[2] School of IT, Deakin University, Australia
leong@deakin.edu.au
[3] Faculty of IT, Monash University, Australia
vincent.cs.lee@monash.edu

Abstract. We consider a CPU constrained environment for finding approximation of frequent sets in data streams using the landmark window. Our algorithm can detect overload situations, i.e., breaching the CPU capacity, and sheds data in the stream to "keep up". This is done within a controlled error threshold by exploiting the Chernoff-bound. Empirical evaluation of the algorithm confirms the feasibility.

1 Introduction

Many new applications now require compute of vast amount of data. The data is fast changing, infinitely sized and has evolving changing characteristics. Data mining them over limited compute resources is thus a problem of interest. As data rates change to exceed compute capacity, the machinery must adapt to produce results on time. The way to achieve this is to incorporate approximation and adaptability into the algorithm.

We introduce an algorithm that addresses approximation and adaptability for finding frequent sets. While this paper reports the method of finding frequent sets in landmark windows, we have in fact developed three methods of finding frequent sets for the landmark window, forgetful window, and sliding window [6]. All three methods are able to adjust their compute according to available CPU capacity and the data rate seen in the stream. All three methods achieved this outcome by shedding load to keep up with the data rate. A significant contribution of our approach is that we imposed an error guarantee on the load shed action using a probabilistic load shedding algorithm. In this paper, we report one of the three algorithms developed.

2 Related Works

The study of load shedding in data streams first begun in query networks as a formulation of query plans, where "drop operations" are inserted [3,7,10]. In data mining models for data streams, those that considers load shedding to

E. Hüllermeier et al. (Eds.): SUM 2012, LNAI 7520, pp. 590–597, 2012.

deal with overload situations include [5] and [2]. Nevertheless, these works do not deal with counting frequent sets. In works addressing frequent sets mining, we can classify them into three categories. Those that count frequent sets in a landmark window, e.g., [12]; those that count frequent sets in a forgetful (or decaying) window, e.g., [4,8]; and those that count frequent sets in a sliding window, e.g., [9]. These techniques all focus on summarising the data stream within the constraints of the main memory. In contrast, our work addressed CPU limits and also provided an error guarantee on the results.

The closest work to what we are proposing is found in [1] where sampling of the *entire* database is performed. Multiple samples are taken as [1] runs in a grid setting. The sampling is controlled via a probability parameter that is not a function of CPU capacity. Further, the approximate results are also not guaranteed by an error limit. Therefore, this contrasts with our proposed algorithm, where we 'sample' only in overload situations and the degree of sampling is controlled by the CPU capacity rather than a pre-set probability parameter. We operate our algorithm on a single compute – not over the grid, and we provide an error guarantee to our results.

3 Algorithm StreamL

We assume a data stream being transactions $\{t_1, t_2, \ldots, t_n, \ldots\}$, where $t_i \subseteq I = \{a_1, a_2, \ldots, a_m\}$ and a_j is a literal to a data, object, or item. We further denote t_n as the current transaction arriving in the stream. A data stream $\{t_1, \ldots\}$ is conceptually partitioned into time slots $TS_0, TS_1, \ldots, TS_{i-k+1}, \ldots, TS_i$, where TS_n holds transactions that arrived at interval (or period) n.

Definition 1. *A **landmark window** is all time slots between the given interval n and the current time slot i, i.e., $TS_n, TS_{n+1}, \ldots, TS_{i-1}, TS_i$. And in terms of transactions, it will be transactions in those time slots. A landmark window hence allows a analyst to perform analytic over a specific period. A data stream of call statistics for example, would benefit from a landmark window where analysis on the past 24 or 48 hours of calls can be obtained. In this case, the last 24 hours would be the landmark window's size.*

Definition 2. *An itemset X is a set of items $X \in (2^I - \{\varnothing\})$. An ℓ-itemset is an itemset that contains exactly ℓ items. Given $X \subseteq I$ and a transaction t_i, t_i supports (or contains) X if and only if $X \subseteq t_i$.*

Definition 3. *Given an itemset X of size ℓ (i.e., an ℓ-itemset), a set of all its subsets, denoted by the power set $\mathcal{P}(X)$, is composed of all possible itemsets that can be generated by one or more items of the itemset X, i.e., $\mathcal{P}(X) = \{Y | Y \in (2^X - \{X\}) \wedge (Y \neq \varnothing)\}$. An immediate subset Y of X is an itemset such that $Y \in \mathcal{P}(X)$ and $|Y| = (\ell - 1)$.*[1]

Definition 4. *The cover of X in $\mathcal{DS}$ is defined as the set of transactions in $\mathcal{DS}$ that support X, i.e., $cover(X, \mathcal{DS}) = \{t_i | t_i \in \mathcal{DS}, X \subseteq t_i\}$.*

[1] $|Y|$ denotes the number of items in Y.

Definition 5. *The frequency of X, denoted by $freq(X)$, is the number of transactions in $\mathcal{DS}$ that contain X, i.e., $freq(X) = |cover(X, \mathcal{DS})|$, and its support, $supp(X)$, is the ratio betweeb $freq(X)$ and $|\mathcal{DS}|$.*

3.1 Workload Estimation

As indicated, our algorithm has two unique characteristics. The first is adaptability to the time-varying characteristics of the stream. In this case, the issue is to detect overload situations. The first instance when an overload occur lie not in detecting the increased rate of transaction arrivals. Rather, it is a combination of arrival rate and the number of literals present in each transaction. We denote the measure reflecting arrival rate and transaction size as L. Under an overloaded situation, figuring the exact L is not a pragmatic solution. Instead we favour an estimation technique for L that is computationally affordable in overloaded situations.

The intuition to estimate L lie in the exponentially small number of maximal frequent sets (MFS) compared to usual frequent sets [11]. Let m be the number of MFS in a transaction t and denote X_i as a MFS, $1 \leqslant i \leqslant m$. Then the time to determine L[2] for each transaction is given as

$$L = \sum_{i=1}^{m} 2^{|X_i|} - \sum_{i,j=1}^{m} 2^{|X_i \bigcap X_j|} \tag{1}$$

and for n transactions over a single time unit, we have the following inequality for load shedding decision

$$P \times r \times \frac{\sum_{i=1}^{n} L_i}{n} \leq C \tag{2}$$

where $P \in (0, 1]$ is repeatedly adjusted to hold the inequality. Hence, the LHS of the inequality also gives the maximum rate for transactions to be processed within a time unit, i.e., at $P = 1$. The process capacity of the system is denoted as C. Hence, the objective is to find all frequent sets while guaranteeing that the workload is always below C.

3.2 Chernoff Bound Load Shedding

Clearly when load is shed, the consequence is the loss of accuracy in the frequency of an itemset. Therefore, it makes sense to put an error guarantee on the results so that analysis can be made with proper context. We achieved this in our algorithm with the Chernoff bound.

Recall that $P = 1$ sets the arrival rate to its maximum. If this breaches C, then we must adjust P for $P < 1$ is when transactions are dropped to meet CPU

[2] Of course, computing L directly will be expensive and impractical. Instead, the implementation uses a prefix tree, where the transaction in question is matched to estimate the number of distinct frequent sets.

constraints. If N are all the transactions, then n represents the sampling of N transactions with $N - n$ the number of transactions dropped. For each itemset X, we compare the frequency for n sampled transactions against the frequency for the original N transactions.

The existence of X in a transaction is a Bernoulli trial that can be represented with a variable A_i such that $A_i = 1$ if X is in the i-th transaction and $A_i = 0$ otherwise. Obviously, $\Pr(A_i = 1) = p$ and $\Pr(A_i = 0) = 1 - p$ with $\Pr(.)$ being the probability of a condition being met. Thus, n randomly drawn transactions are n independent Bernoulli trials. Further, let r (a *binomial* random variable) be the occurrence of $A_i = 1$ in the n transactions with expectation np. Then, the Chernoff bound states that for any $\varepsilon > 0$ and $\epsilon = p\varepsilon$, we have

$$\Pr\{|r - np| \geq n\epsilon\} \leq 2e^{-n\epsilon^2/2p} \tag{3}$$

If $s_E(X) = r/n$ is the estimated support of X computed from n sampled transactions, then the Chernoff Bound tells us how likely the true support $s_T(X)$ deviates from its estimated support $s_E(X)$ by $\pm\epsilon$. If we want this probability to be $\leqslant \delta$, the sampling size should be at least (by setting $\delta = 2e^{-n\epsilon^2/2p}$)

$$n_0 = \frac{2p\ln(2/\delta)}{\epsilon^2} \tag{4}$$

3.3 Algorithmic Steps

In landmark windows, we compute answers from a specific time in the past to the latest transaction seen. For ease of exposition, we assume the first transaction within a landmark window to be t_1. Hence, we can conceptually compute as if the entire stream is seen, i.e., dropping the time boundary. Further, we *conceptually* divide the stream into buckets; each holding Δ transactions. To keep results within a given threshold, we set $\Delta = n_0$ (Equation. 4).

Due to resource limits, we do not count all itemsets. Rather, we identify the candidates that will become frequent and track those. With the conceptual buckets, we use a deterministic threshold $\gamma(< \sigma)$ to filter unlikely frequent sets. If an itemset's support falls below γ, then it is unlikely to become frequent in the near future. Since an approximate collection of frequent sets are produced, this must be within the promised error guarantee. As transactions are only read once, there is a higher error margin with longer frequent sets as a potential candidate is only generated after all its subsets are significantly frequent, i.e., the downward closure property.

To counter this, an array that tracks the minimum frequency thresholds is used to tighten the upper error bound. We set m to the longest frequent sets of interest to the user. The array thus must satisfy $a[i] < a[j]$ for $1 \leq i < j \leq m$, $a[1] = 0$ (for 1-itemsets) and $a[m] \leq \gamma \times \Delta$. A j-itemset is thus generated if its immediate subsets in the current bucket is $\geqslant a[j]$. During load shedding, we set each bucket to Δ transactions to bound the true support error probability to no more than ϵ. As will be discussed next, the support error of frequent sets discovered is guaranteed to be within γ and ϵ.

The StreamL algorithm uses a prefix tree $\mathcal{S}$ to maintain frequent sets discovered from the landmark window. Each node in $\mathcal{S}$ corresponds to an itemset X represented by items on a path from the root to the node, where $Item$ is last literal in X; $f_c(X)$ is the frequency of X in the current bucket; and $f_a(X)$ is the accumulated frequency of X in the stream.

Further, let b_c be the most current bucket in the stream. Hence, $b_c = 1$ upon initialisation and is incremented each time Δ transactions are processed. The algorithmic steps of StreamL is thus given below.

1. Periodically detect capacity C and if no overload, set $P = 1$. Otherwise, set P to maximal value identified in Equation 2.
2. Process transaction t_n with probability p as follows
   Increment if $X \subseteq t_n$ is in $\mathcal{S}$, then $f_c(X) = f_c(X) + 1$
   Insert, where $|X| = 1$ $\forall X \subseteq t_n \wedge X \notin \mathcal{S} \rightarrow$ insert X into $\mathcal{S}$ with $b(X) = b_c$ and $f_c(X) = 1$[3];
   Insert, where $|X| > 1$ Insert X into $\mathcal{S}$ when *all* following conditions hold
   - All immediate subsets of X are in $\mathcal{S}$;
   - $\nexists Y$ (Y is any immediate subsets of X) *s.t.* $b(Y) < b_c$ and $f_c(Y) \leq a[|X|]$, i.e., no Y is inserted into $\mathcal{S}$ from previous buckets but its frequency in b_c is insufficiently significant;
   - $\nexists Y$ *s.t.* $b(Y) = b_c$ and $f_c(Y) \leq (a[|X|] - a[|Y|])$, i.e., no Y is inserted into $\mathcal{S}$ in b_c and after which its frequency is $\leqslant (a[|X|] - a[|Y|])$.

   In cases where X is not inserted into $\mathcal{S}$, all its supersets in t_n need not be further checked.
   Prune $\mathcal{S}$ and update frequency After sampling Δ transactions, remove all itemsets (except 1-itemsets) in $\mathcal{S}$ fulfilling $f_a(X) + f_c(X) + b(X) \times a[|X|] \leq b_c \times a[|X|]$. If an itemset is removed, all its supersets are also removed. This does not affect itemsets insert into bucket b_c since their immediate subsets have became sufficiently frequent. For unpruned itemsets, their frequencies are updated as $f_a(X) = f_a(X) + f_c(X) \times P^{-1}$ and $f_c(X) = 0$. To compensate the effects of dropping transactions, X's frequency is scaled by P^{-1} to approximate its true frequency in the data stream. Finally, we update $b_c = n/\Delta + 1$.
3. At any point when results are requested, StreamL scans $\mathcal{S}$ to return 1-itemsets satisfying $f_a(X) + f_c(X) \times P^{-1} \geqslant \sigma \times n$ and ℓ-itemsets satisfying $f_a(X) + f_c(X) \times P^{-1} + b(X) \times a[|X|] \geqslant \sigma \times n$.

3.4 Error Analysis

We now present the accuracy analysis on the frequency estimates produced by our algorithm. For each itemset X[4], we denote its *true* frequency by $f_T(X)$ and

[3] The immediate subsets of a 1-itemset is $\varnothing$ which appears in every transaction. All 1-itemsets are therefore inserted into $\mathcal{S}$ without condition. For the same reason, they are also not pruned from $\mathcal{S}$.

[4] We note that we will not discuss the error for 1-itemsets as their support will be precisely counted in the absence of load shedding. Therefore the true and estimated support are the same for these itemsets.

estimated frequency by $f_E(X) = f_a(X) + f_c(X)$ using synopsis $\mathcal{S}$. Respectively, $s_T(X)$ and $s_E(X)$ denote its *true* and *estimated* supports.

Lemma 1. *Under no load shedding (i.e., $P = 1$), StreamL guarantees if X is deleted at b_c, its true frequency $f_T(X)$ seen so far is $\leqslant b_c \times a[|X|]$.*

Proof. This lemma can be proved by induction. Base case when $b_c = 1$: nothing is deleted at the end of this bucket since an itemset is inserted into $\mathcal{S}$ only if its immediate subsets are sufficiently significant. Therefore, any itemset X not inserted into $\mathcal{S}$ in the first bucket will not have their true frequency greater than $a[|X|]$; i.e., $freq_T(X) \leqslant a[|X|]$. Note that the maximal error in counting frequency of an inserted itemset X (at this first bucket) is also equal to $a[|X|]$. This is because X is inserted into $\mathcal{S}$ and started counting frequency *only after* all X's subsets gain enough $a[|X|]$ on their occurrences.

When $b_c = 2$: Likewise, itemsets inserted into this bucket are not deleted since they are in $\mathcal{S}$ only after their subsets are confirmed significant. Only itemsets inserted in the first bucket might be removed as they can become infrequent. Assume that X is such itemset, its error is at most $a[|X|]$. On the other hand, $f_E(X)$ is its frequency count since being inserted in the first bucket. Thus, $f_T(X) \leqslant f_E(X) + a[|X|]$. Combing with the delete condition: $f_E(X) + a[|X|] \leqslant b_c \times a[|X|]$, we derive $f_T(X) \leqslant b_c \times a[|X|]$.

Induction case: Suppose X is deleted at $b_c > 2$. Then, X must be inserted at some bucket $b_i + 1$ before b_c; i.e., $b(X) = b_i + 1$. In the worst case, X could possibly be deleted in the previous bucket b_i. By induction, $f_T(X) \leqslant b_i \times a[|X|]$ when it was deleted in b_i. Since $a[|X|]$ is the maximum error at b_i+1 and $f_E(X)$ is its frequency count since being inserted in b_i+1, it follows that $f_T(X)$ is at most $f_E(X)+b_i \times a[|X|]+a[|X|] = f_E(X)+(b_i+1)\times a[|X|]$; or $f_T(X) \leqslant f_E(X)+b(X)\times a[|X|]$. When combined with the deletion rule $f_E(X)+b(X)\times a[|X|] \leqslant b_c \times a[|X|]$, we get $f_T(X) \leqslant b_c \times a[|X|]$.

Theorem 1. *Without load shedding, StreamL guarantees that the true support of any $X \in \mathcal{S}$ is limited within the range: $\mathrm{s}_E(X) \leqslant \mathrm{s}_T(X) \leqslant \mathrm{s}_E(X) + \gamma$.*

Proof. By definition $f_E(X)$ is the counting of X since it was inserted into $\mathcal{S}$. Therefore the true frequency of X is always at least equal to this value, i.e., $f_E(X) \leq f_T(X)$. We now prove that $f_T(X) \leq f_E(X) + b(X) \times a[|X|]$.

If X is inserted in the first bucket, its maximal frequency error is $a[|X|]$. Therefore, $f_T(X) \leqslant f_E(X) + a[|X|]$; In the other case, X is possibly deleted some time earlier in the first b_i buckets and then inserted into $\mathcal{S}$ at $b_i + 1$. By Lemma 1, $f_T(X)$ is at most $b_i \times a[|X|]$ when such a deletion took place. Therefore, $f_T(X) \leqslant f_E(X) + a[|X|] + b_i \times a[|X|] = f_E(X) + b(X) \times a[|X|]$. Together with the result above, we derive $f_E(X) \leqslant f_T(X) \leqslant f_E(X) + b(X) \times a[|X|]$. On the other hand, we define $a[i] < a[j]$ for $1 \leq i < j \leq m$ and $a[m] \leq \Delta\gamma$. And since $\Delta \times b(X)$ is always smaller than the number of transactions seen so far in the stream, i.e., $\Delta \times b(X) \leq n$, we have $a[|X|] \times b(X) \leq n \times \gamma$. Dividing the inequality above for n, the theorem is proven.

Theorem 2. *With load shedding, StreamL guarantees that* $s_E(X) - \epsilon \leq s_T(X) \leq s_E(X) + \gamma + \epsilon$ *with a probability of at least* $1 - \delta$.

Proof. This theorem can be directly derived from the Chernoff bound and Theorems 1. At periods where the load shedding happens, the true support of X is guaranteed within $\pm\epsilon$ of the counting support with probability $1 - \delta$ when the Chernoff bound is applied for sampling. Meanwhile, by Theorems 1, this counting support is limited by $s_E(X) \leq s_T(X) \leq s_E(X) + \gamma$. Therefore, getting the lower bound in the Chernoff bound for the left inequality and the upper bound for the right inequality, we derive the true support of X to within the range $s_E(X) - \epsilon \leq s_T(X) \leq s_E(X) + \gamma + \epsilon$ with a confidence of $1 - \delta$.

4 Empirical Evaluation

We summarise our results here but refer the reader to our technical reports for an in-depth discussion to meet space constraints.

We used the IBM QUEST synthetic data generator for our test. The first data set `T10.I6.D3M` has an averaged transaction length of $\mathtt{T} = 10$, average frequent set length of $I = 6$ and contains 3 million transactions (D). The other two data sets are `T8.I4.D3M` and `T5.I3.D3M`. On all three datasets, we are interested in a maximum of $L = 2000$ potentially frequent sets with $N = 10,000$ unique items. As our algorithm is probabilistic, we measure the *recall* and *precision* on finding the the true frequent sets. The experiment is repeated 10 times for each parameter setting to get the average reading.

To test **accuracy**, we assume a fixed CPU capacity and varied the load shed percentage between 0% and 80%. A shed load of 50% for example corresponds to an input stream rate that is twice the CPU capacity. When $P = 1$, the algorithm behaves like its deterministic cousins with recall of 100% but its precision is not 100%. This lack of precision is due to false positives as a result of over estimating true occurrences by $\gamma \times n_w$. When load is shed, the false positives and false negatives increase as reflected in the precision and recall. This is as expected since less transactions are processed during a load shed and the action impacts accuracy. The important point though is that the accuracy of the results across our experiments show that it is highly maintained – in two of the three data sets, both measures were above 93%. We also observed that as load sheds increases beyond 60%, there is a general huge drop of accuracy so we note the practical extend of shedding possible.

To test the **adaptability** of our algorithm, we create a hybrid data set by concatenating the first 1 million transactions from `T5.I3.D3M`, `T8.I4.D3M` and `T10.I6.D3M` respectively. We note from the experiments that the time to compute the statistics for managing CPU capacity is negligible compared to finding frequent sets. The cost of maintaining the statistics is also linearly proportioned to the time across support thresholds experimented. From the results seen, we can conclude that we can use the statistics collected to identify the appropriate

amount of data for shedding. On the results seen, the adaptability of our algorithm is practically reasonable as the cost to achieve a response to the load is relatively small.

5 Conclusions

We considered computing frequent sets in streams under unpredictable data rates and data characteristics that affect CPU capacity. Our unique contribution is a parameter to control the stream load imposed on CPU capacity through load shedding. Additionally, we bound the error to a Chernoff bound to ensure that the error is guaranteed to within a certain threshold.

References

1. Appice, A., Ceci, M., Turi, A., Malerba, D.: A Parallel Distributed Algorithm for Relational Frequent Pattern Discovery from Very Large Data Sets. Intelligent Data Analysis 15(1), 69–88 (2011)
2. Bai, Y., Wang, H., Zaniolo, C.: Load Shedding in Classifying Multi-Source Streaming Data: A Bayes Risk Approach. In: SDM (2007)
3. Babcock, B., Datar, M., Motwani, R.: Load Shedding for Aggregation Queries over Data Streams. In: ICDE, pp. 350–361 (2004)
4. Chang, J.H., Lee, W.S.: Finding Recent Frequent Itemsets Adaptively over Online Data streams. In: SIGKDD (2003)
5. Chi, Y., Yu, P.S., Wang, H., Muntz, R.R.: LoadStar: A Load Shedding Scheme for Classifying Data Streams. In: SDM, pp. 346–357 (2005)
6. Dang, X., Ong, K.-L., Lee, V.C.S.: Real-Time Mining of Approximate Frequent Sets over Data Streams Using Load Shedding. Deakin University, Technical Report (TR 11/06) (2011), `http://www.deakin.edu.au/~leong/papers/dol.pdf`
7. Gedik, B., Wu, K.-L., Yu, P.S., Liu, L.: Adaptive Load Shedding for Windowed Stream Joins. In: CIKM, pp. 171–178 (2005)
8. Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.S.: Mining Frequent Patterns in Data Streams at Multiple Time Granularities. Next Generation Data Mining. AAAI/MIT (2003)
9. Lin, C., Chiu, D., Wu, Y., Chen, A.: Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window. In: SDM, pp. 68–79 (2005)
10. Tatbul, N., Çetintemel, U., Zdonik, S.B.: Staying Fit: Efficient Load Shedding Techniques for Distributed Stream Processing. In: VLDB, pp. 159–170 (2007)
11. Yang, G.: The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns. In: SIGKDD, pp. 344–353 (2004)
12. Yu, J.X., Chong, Z., Lu, H., Zhou, A.: False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In: VLDB, pp. 204–215 (2004)