In-stream Frequent Itemset Mining with Output Proportional Memory Footprint

Daniel Trabold[1], Mario Boley[2], Michael Mock[1], and Tamas Horváth[2,1]

[1]Fraunhofer IAIS, Schloss Birlinghoven, 53754 St. Augustin, Germany
[2]Dept. of Computer Science, University of Bonn, Germany
{daniel.trabold,michael.mock,tamas.horvat}@iais.fraunhofer.de
mario.boley@gmail.com

Abstract. We propose an online partial counting algorithm based on statistical inference that approximates itemset frequencies from data streams. The space complexity of our algorithm is proportional to the number of frequent itemsets in the stream at any time. Furthermore, the longer an itemset is frequent the closer is the approximation to its frequency, implying that the results become more precise as the stream evolves. We empirically compare our approach in terms of correctness and memory footprint to CARMA and Lossy Counting. Though our algorithm outperforms only CARMA in correctness, it requires much less space than both of these algorithms providing an alternative to Lossy Counting when the memory available is limited.

1 Introduction

Mining frequent itemsets from data streams with small memory footprint is an important data mining task with many applications such as, for example, credit card payment monitoring or phone call processing A stream setting requires algorithms that provide aggregated results on the current status of the stream in low latency at any time. Furthermore, the memory consumption must remain small, ideally proportional to the number of frequent itemsets in the input. A small overhead per transaction may add up to a huge amount for long streams.

The goal of itemset frequency estimation is to report the frequencies for all itemsets which occur more frequent than some threshold in a stream of transactions. A number of papers consider the simpler problem of counting single items or 1-itemsets from streams (see, e.g., [1–3]). All known solutions for the task of itemset frequency estimation fall in one of the following two categories: Algorithms without buffers and algorithms with some form of transaction buffering. The former include CARMA [4] and HMINER [5].

CARMA determines a superset of all frequent itemsets in a first pass and needs a second pass to compute the exact family of frequent itemsets. Requiring a

 In: R. Bergmann, S. Gorg, G. Muller (Eds.): Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB. Trier, Germany, 7.-9. October 2015, published at http://ceur-ws.org

second pass is, however, unrealistic for large data streams. In contrast to CARMA, our algorithm does not require a second pass. Regarding hMINER, it has the drawback that the maximum number of possible items in the stream must be known for the algorithm in advance. In contrast to hMINER, we do not assume any such prior knowledge.

One of the representative algorithms belonging to the second category is the Lossy Counting algorithm [2], which has a frequency threshold θ and an error parameter ϵ with $\epsilon < \theta$. This algorithm stores not only the itemsets with frequency at least θ, but also all itemsets with frequency above ϵ, implying that Lossy Counting counts also a large number of infrequent itemsets with frequency between ϵ and θ and keeps those counts in the memory.

This paper presents an algorithm that, at any point in time, approximates the family of frequent itemsets and their support counts from a data stream. The central idea of our algorithm is to combine partial counts with a simple, yet sound statistical model of inference. Our algorithm starts counting an itemset when all of its non-empty proper subsets have reached the minimal frequency threshold and prunes them immediately when they are no longer frequent. Because counting starts only when all subsets are frequent, we may have only partial counts even for the frequent itemsets. To obtain the frequency for the entire stream, our algorithm uses statistical inference to explicitly model the probabilities of itemsets observed from a certain time point t and estimates the frequency for the unobserved interval $[1, t-1]$. Each transaction contributes evidence to this model, making our approach more accurate. In fact, one can show that for infinite data streams, our algorithm is always correct in the limit.

We have compared our algorithm to the CARMA [4] and LOSSY COUNTING [2] algorithms. In our experimental evaluations we have measured the accuracy and the memory footprint for all of the three algorithms, where the accuracy has been measured by calculating the Jaccard distance between the set of correct frequent itemsets and that of the output of the algorithms. While our algorithm has outperformed CARMA both in accuracy and memory, LOSSY COUNTING has generated the most accurate output. However, regarding the memory consumption, our algorithm has required much less space than LOSSY COUNTING. Thus, once the space available is limited, our algorithm can be regarded as an alternative to LOSSY COUNTING.

The rest of the paper is organized as follows. We first collect the necessary preliminaries in Section 2 and then present our algorithm in Section 3. In Section 4 we discuss some estimation strategies natural for our algorithm. Section 5 provides some important details about the implementation, while Section 6 presents our empirical results. We conclude in Section 7 with some interesting problems for future works.

2 Preliminaries

We follow the standard terminology used in frequent itemset mining (see, e.g., [6, 7]). In particular, for a set I of items, a subset $X \subseteq I$ will be referred to as *itemset*.

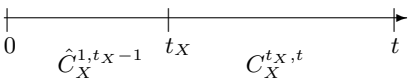


Fig. 1. Transaction stream: The occurrences of itemset X are counted from t_X. $C_X^{t_X,t}$ denotes this observed count for X from t_X to t. As the support count of X is not available for the period 1 to $t_X - 1$, it must be estimated for this time interval; $\hat{C}_X^{1,t_X-1}$ denotes this estimated count.

If for the cardinality of X we have $|X| = k$ then X is a *k-itemset*. We consider a potentially infinite stream of transactions, i.e., a potentially unbounded sequence of subsets of I, that are observed consecutively at discrete time points. Moreover, we assume that each element of this stream is generated independently according to some *fixed*[1], but *unknown* distribution $\mathcal{D} : 2^I \to [0, 1]$. Formally, we have an input sequence $D_1, D_2, ...$ with $D_t \subseteq I$ and $D_t \sim \mathcal{D}$ for each point in time $t \in \mathbb{N}$.

For an itemset $X \subseteq I$ we define the *support count* of X from time i to j by

$$C_X^{i,j} = |\{t \in \mathbb{N} : i \leq t \leq j, X \subseteq D_t\}| \ ,$$

i.e., the support count of X is equal to the number of transactions from D_i to D_j that contain X. We define the *frequency* of X at time t as the relative support count

$$\text{freq}_t(X) = \frac{C_X^{1,t}}{t} \ .$$

Finally, using these concepts, the family of frequent sets at time t with respect to a frequency threshold $\theta \in [0, 1]$ is given by

$$\mathcal{F}_{t,\theta} = \{X \subseteq I : \text{freq}_t(X) > \theta\} \ .$$

Our goal in this paper is to design an algorithm that produces a sequence $\mathcal{C}_1, \mathcal{C}_2, ...$ of families of itemsets such that at each point in time $t \in \mathbb{N}$, the family $\mathcal{C}_t$ approximates $\mathcal{F}_{t,\theta}$ closely, while maintaining a small memory footprint. To evaluate the approximation performance of our and other algorithms, we will use the Jaccard distance as loss function defined for all $A, B \subseteq 2^I$ by

$$l(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \ .$$

3 Estimating support counts based on partial counts

In this section we present our Partial Counting algorithm approximating the frequency of frequent itemsets from a data stream of transactions at any point of time. The inputs to the algorithm are a minimum frequency threshold $\theta \in [0, 1]$ and, for each time point $t \in \mathbb{N}$, a transaction $D_t \subseteq I$. For each

[1] In the long version of this paper we will show how to get rid of this assumption.

$t \in \mathbb{N}$, the algorithm estimates the frequency of all frequent itemsets with respect to the transaction database $\{D_1, \ldots, D_t\}$, in one pass and without storing the transactions $D_1, \ldots, D_{t-1}$ seen earlier.

For all $t \in \mathbb{N}$, the algorithm keeps only those itemsets in the memory that have been estimated as frequent after having processed D_t; all other itemsets, except for the 1-itemsets, are removed from the memory. Thus, in contrast to e.g. the LOSSY COUNTING algorithm [2], the space complexity of our algorithm is only $O(\hat{\mathcal{F}}_{t,\theta})$, where $\hat{\mathcal{F}}_{t,\theta}$ is the family of itemsets estimated as frequent with respect to $D_1, \ldots, D_t$.

The algorithm is depicted in Algorithm 1. To process the next transaction D_t arriving at time t, the algorithm takes in a first step (Lines 4–12) all non-empty subsets X of D_t (Line 4) and increments the counter for X if X is already in the memory (Lines 5–6). Otherwise, if X is a singleton or it is a k-itemset for some $k > 1$ and all of its $k-1$-subsets are already in the memory, it stores X with some additional auxiliary information. We utilize the Apriori property, i.e., that a k-itemset cannot be frequent if at least one of its (k-1)-subsets is infrequent. More precisely, for X we store a quadruple $(x.set, x.s, x.t, x.count)$ that will be used to estimate the frequency of X for the data stream $D_1, \ldots, D_{t-1}$ (see Lines 8–12 for the definitions of entries in the quadruple). It is important to note that the subsets of D_t are processed in increasing cardinalities, as otherwise potential new frequent itemsets can be lost (see Lines 4,8, and 9).

In a second step (Lines 13–16) the algorithm then prunes all quadruples corresponding to itemsets X from the memory that satisfy $|X| > 1$ and are estimated as infrequent at point t. In Line 14, we first calculate an estimation of the support count of X; we present different strategies for this estimation step in Section 4 by noting that all these strategies estimate the support count from the counts (i.e., $x.count$) maintained by the algorithm using some statistical inference. Figure 1 illustrates the general setting: For an itemset X (re)counted from time t_X, its support count for the period from 1 to $t_X - 1$ must be estimated from the information available at time t. If the frequency derived from this estimation is below the threshold θ, X is removed from the memory. Thus, when an itemset becomes frequent it is stored and counted as long as it is estimated as frequent. When it becomes infrequent, it is immediately pruned.

For a query after time point t, we output all itemsets with their estimated support counts from $\mathcal{F}$ that meet the minimum frequency condition (Line 18–19). According to the construction, all itemsets X in $\mathcal{F}$ with $|X| > 1$ will automatically be part of the output. In summary, we have a true online algorithm that returns a family $\hat{\mathcal{F}}_{t,\theta}$ of itemsets predicted as frequent from the stream of transactions from the beginning of the stream up to time t.

4 Support count estimators

In this section we describe a generic framework for support count estimation and present different strategies for this problem. Except for one, all of the strategies

Algorithm 1 Partial Counting

Intitalization
$\mathcal{F} \leftarrow \emptyset$ // current set of frequent patterns with auxiliary information

Processing of transaction D_t
for $X \subseteq D_t$ in increasing cardinality **do** // counting
  if $\exists x \in \mathcal{F}$ with $x.set = X$ **then**
    increment $x.count$
  else
    if $|X| = 1 \vee (\forall Y \subset X : \exists y \in \mathcal{F}$ with $y.set = Y \wedge |Y| = |X| - 1)$ **then**
      $x.s \leftarrow \{(y.set, y.count) : y \in \mathcal{F} \wedge y.set \subset x.set \wedge |y.set| = |x.set| - 1)\}$
      $x.t = t$
      $x.count = 1$
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{x\}$
for $x \in \mathcal{F}$ with $|x.set| > 1$ **do** // pruning
  compute $\hat{C}_x^{1,t}$ // see Section 4
  if $\hat{C}_x^{1,t}/t < \theta$ **then**
    delete x from $\mathcal{F}$

Output after timepoint t
for $x \in \mathcal{F}$ **do**
  if $|x.set| > 1 \vee x.count/t > \theta$ **then** output (x.set,$\hat{C}_x^{1,t}$)

in this section are based on some careful combination of the observed counts with the estimated ones that are derived from conditional probabilities.

We write $\hat{C}$ for estimated support counts. As illustrated in Figure 1, whenever there is some observed support count for an itemset X, the estimated support count for the entire period of all transactions is given by the *observed* support count $C_X^{t_X,t}$ plus the *estimation* $\hat{C}_X^{1,t_X-1}$ of the support count for the time period $[1, t_X - 1]$ for which the support count of X is not available at point t. As long as no observed count for X exists, its estimated frequency is 0, i.e.,

$$\hat{C}_X^{1,t} = \begin{cases} \hat{C}_X^{1,t_X-1} + C_X^{t_X,t} & \text{if } t \geq t_X \\ 0 & \text{o/w .} \end{cases}$$

We now present different strategies to compute $\hat{C}_X^{1,t_X-1}$. We first recall an estimation from [4] and then propose two natural strategies based on conditional probabilities.

4.1 Upper bound estimation (ube)

This estimation strategy is used in CARMA [4]. It takes the minimum count of all $(k-1)$-subitemsets Y of a k-itemset X, i.e.,

$$\hat{C}_X^{1,t_X-1} = \min_{\substack{Y \subset X, \\ |Y|=|X|-1}} \hat{C}_Y^{1,t_X-1} \ .$$

Clearly, this formula gives an upper bound on the true support count of X. Notice that this is a static strategy in the sense that it does not improve the estimation $\hat{C}_X^{1,t_X-1}$ as further transactions arrive.

4.2 Estimation based on conditional probabilities

We now turn to more complex, dynamic estimation strategies. They are based on the probabilistic view of frequencies that for any itemset X and $t \in \mathbb{N}$

$$p(X)^{1,t} = p(Y)^{1,t} \cdot p(X|Y)^{1,t}$$

for any $Y \subset X$. To estimate $p(X)^{1,t}$, we need to estimate $p(X)^{1,t_X-1}$, as all information about X is available for the interval $[t_X, t]$. We estimate $p(X)^{1,t_x-1}$ by estimating (i) $p(Y)^{1,t_X-1}$ and (ii) $p(X|Y)^{1,t_X-1}$.

(i) Regarding $p(Y)^{1,t_X-1}$, we estimate it recursively by

$$p(Y)^{1,t_X-1} \approx \frac{\hat{C}_Y^{1,t_X-1}}{t_X - 1} \tag{1}$$

by noting that the support counts are stored from the very beginning for all 1-itemsets.

(ii) Regarding $p(X|Y)^{1,t_X-1}$, we make use of the fact that $[t_X, t]$ is a common observation period for both X and Y and the assumption that the underlying distribution $\mathcal{D}$ is stationary and estimate $p(X|Y)^{1,t_X-1}$ by

$$p(X|Y)^{1,t_X-1} \approx p(X|Y)^{t_X,t}, \tag{2}$$

which, in turn, can be calculated by

$$p(X|Y)^{t_X,t} = \frac{C_X^{t_X,t}}{C_Y^{t_X,t}} \ . \tag{3}$$

One can show that for sufficiently large t_X and t, (2) gives a close estimation with high probability.

Putting together, from (1), (2), and (3) it follows that $p(X)^{1,t_X-1}$ can be estimated by

$$p(X)^{1,t_X-1} \approx \frac{\hat{C}_Y^{1,t_X-1}}{t_X - 1} \cdot \frac{C_X^{t_X,t}}{C_Y^{t_X,t}} \ . \tag{4}$$

As the frequency of X in $[1, t_X - 1]$ is identical to the probability $p(X)^{1,t_X-1}$, by (4) the support count $\hat{C}_X^{1,t_X-1}$ can be estimated by

$$\begin{aligned} \hat{C}_X^{1,t_X-1} &= p(X)^{1,t_X-1} \cdot (t_X - 1) \\ &\approx \frac{\hat{C}_Y^{1,t_X-1} \cdot C_X^{t_X,t}}{C_Y^{t_X,t}} \ . \end{aligned}$$

The two strategies presented below build upon the idea discussed above. They differ from each other only by the particular choice of Y. All strategies have in common that they estimate the support counts only for itemsets X with $|X| \geq 2$.

transactions	a	b	a	ab	a	b	a	ab
t	1	2	3	4	5	6	7	8
freq(a)	1	0.5	0.66	0.75	0.8	0.6	0.71	0.75
freq(b)	0	0.5	0.33	0.5	0.4	0.5	0.43	0.5
freq(ab)	0	0	0	0.25	0.2	0.17	0.14	0.25
ube	0	0	0	0.5	0.4	0.33	0.29	0.38
me	0	0	0	0.5	0.4	0.25	0.21	0.33
ae	0	0	0	0.5	0.4	0.29	0.23	0.35

Table 1. A data stream of transactions illustrating the different estimation strategies ube, me, and ae. The first row shows the transactions, the second row t, the next three rows the frequencies for itemsets a, b and ab respectively. The last three rows show the estimated frequencies for the itemsets a, b, and ab for the three strategies presented.

(a) Minimum estimation (me): This strategy uses the single subset Y that results in the minimum estimated count for X, i.e.,

$$\hat{C}_X^{1,t_X-1} = \begin{cases} \min\limits_{\substack{Y \subset X, \\ |Y|=|X|-1}} \dfrac{\hat{C}_Y^{1,t_X-1} \cdot C_X^{t_X,t}}{C_Y^{t_X,t}} & \text{if } |X| > 1 \\ 0 & \text{o/w (i.e., if } |X| = 1) \ . \end{cases}$$

(b) Average estimation (ae): Averaging is a standard technique to combine several uncertain predictors for obtaining a more robust result than any individual predictor would give. This strategy averages over all y, i.e.,

$$\hat{C}_X^{1,t_X-1} = \begin{cases} \dfrac{1}{|X|} \cdot \sum\limits_{\substack{Y \subset X, \\ |Y|=|X|-1}} \dfrac{\hat{C}_Y^{1,t_X-1} \cdot C_X^{t_X,t}}{C_Y^{t_X,t}} & \text{if } |X| > 1 \\ 0 & \text{o/w (i.e., if } |X| = 1) \ . \end{cases}$$

4.3 An Illustrative Example

To illustrate the three different strategies presented above, we use the small example given in Table 1. It shows a total of 8 transactions in the first row, t in the second row, and the frequencies of a, b, and ab in rows three to five. The last three rows show the estimated frequencies for the three strategies.

In the example the first estimation occurs for transaction 4 as the set ab occurs for the first time in transaction 4 and is counted from this transaction onwards. Up to and including the third transaction the set a occurs twice, the set b once. All estimations rely on the counts of a and b for the first three transactions. The strategies start to differ from the 6th transaction onwards. We will therefore illustrate the different strategies for the 6th transaction. $C_{ab}^{4,6} = 1$ for all strategies.

(i) ube estimates $\hat{C}_{ab}^{1,3} = min(2, 1) = 1$, which gives

$$\hat{C}_{ab}^{1,6} = \hat{C}_{ab}^{1,3} + C_{ab}^{4,6} = 1 + 1 = 2$$

and thus, a frequency of $\frac{2}{6} = 0.33$.

(ii) me estimates $\hat{C}_{ab}^{1,3} = min(1 \cdot \frac{1}{2}, 2 \cdot \frac{1}{2}) = 0.5$, which gives

$$\hat{C}_{ab}^{1,6} = \hat{C}_{ab}^{1,3} + C_{ab}^{4,6} = 0.5 + 1 = 1.5$$

and a frequency of $\frac{1.5}{6} = 0.25$.

(iii) ae estimates $\hat{C}_{ab}^{1,3} = \frac{1}{2}(1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2}) = 0.75$, which gives

$$\hat{C}_{ab}^{1,6} = \hat{C}_{ab}^{1,3} + C_{ab}^{4,6} = 0.75 + 1 = 1.75$$

and a frequency of $\frac{1.75}{6} = 0.29$.

All other estimations are computed accordingly.

5 Implementation

In this section we briefly discuss some important implementation issues of the Partial Counting algorithm, in particular, the data structure, insertion into the data structure, and pruning.

The data structure $\mathcal{F}$ can be stored as a prefix tree. This allows for a compact representation of the family of itemsets, as it suffices to store for each itemset X only a single item. Indeed, the itemset corresponding to a node can uniquely be recovered by concatenating the items on the path from the root to the node at hand. Furthermore, it also allows for pruning entire branches, if a node has to be deleted which has further children.

The set $x.s$ can be stored compactly as an array of integers. For an itemset X with $|X| = k$, there are k different $(k-1)$-subsets. We sort all these subsets Y lexicographically and take only the index of the missing item i (i.e., which satisfies $Y \cup \{i\} = X$) to store the count for itemset Y. Thus, in this way there are k counters for the subsets and one for the itemset X.

Theoretically, we may start observing an itemset X as soon as the estimated frequencies for all subsets $Y \subset X$ have reached the minimum frequency threshold θ. X may not occur in the transaction, when this condition is met. However, X may become frequent only when it occurs in the current transaction. That is, we start counting X, with the transaction containing X after all Y have already reached the minimum frequency threshold. The price we pay for this is that the first estimation of $p(X|Y)$ is 1 and as such, very inaccurate.

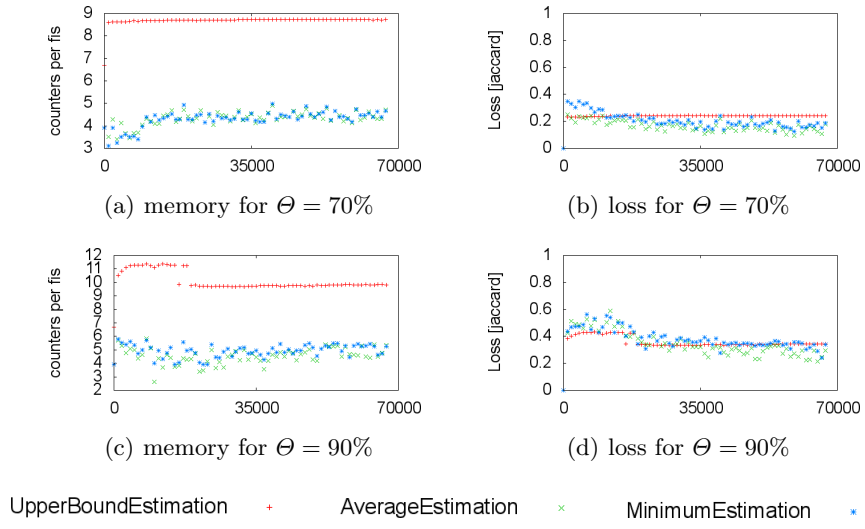
An itemset which is inserted in $\mathcal{F}$ at time t is never pruned in t, otherwise it would not have been inserted. We exploit this by skipping the pruning test for itemsets which were inserted in the same transaction.

6 Experimental Evaluation

To assess the performance of our algorithm in practice, we analyzed the number of counters in memory and the empirical loss for real-world datasets taken from the UCI machine learning repository [8]. The empirical evaluations focus on the following three main aspects:

– Comparing the memory and loss of the different estimation strategies.
– Comparing Partial Counting with Lossy Counting and Carma in terms of memory requirements and loss.
– Evaluating the memory and loss of Partial Counting with decreasing frequency threshold.

For each dataset we shuffle the transactions, add them one by one and mine at regular intervals. The ground truth is obtained with Apriori [6]. We compute the Jaccard distance based loss for each algorithm as defined in Section 2. This loss accounts for both over and underestimation of itemset frequencies without further distinguishing between the two.



(a) memory for $\Theta = 70\%$ (b) loss for $\Theta = 70\%$

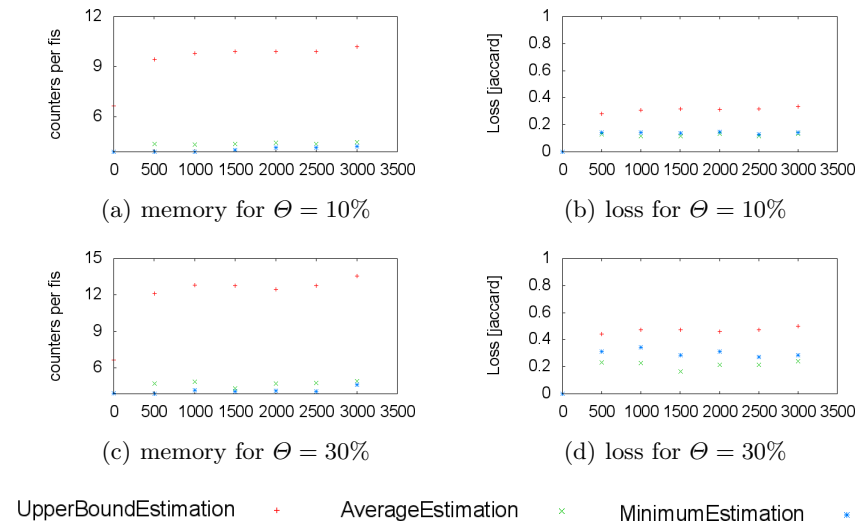
(c) memory for $\Theta = 90\%$ (d) loss for $\Theta = 90\%$

Fig. 2. Memory and loss of Partial Counting with different estimation strategies for dataset connect-4 at 70% (top) and 90% (bottom) frequency threshold.

Overall, the results show that

(i) our algorithm requires less memory than CARMA and Lossy Counting (see Figure 4),
(ii) mining at lower thresholds results in a better approximation (see Figures 2, 3, and 4),
(iii) the two strategies (me and ae) based on statistical inference outperform the simpler strategy (ube) (see Figures 3(b) and 3(d)), and
(iv) the loss decreases as the stream evolves (Figures 2(d) and 4(b)).

We report the memory requirements and loss for the dataset connect-4 with a maximal itemset length of 3 and a minimum frequency threshold of 70% and 90% in Figure 2 and for the chess dataset at 10% and 30% frequency threshold in Figure 3.



(a) memory for $\Theta = 10\%$ (b) loss for $\Theta = 10\%$

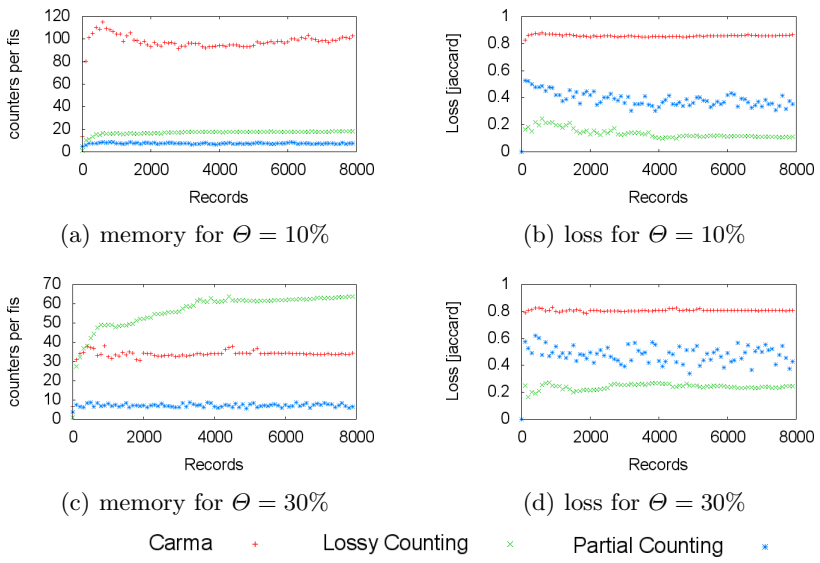
(c) memory for $\Theta = 30\%$ (d) loss for $\Theta = 30\%$

Fig. 3. Memory and loss of Partial Counting with different estimation strategies for dataset chess at 10% (top) and 30% (bottom) frequency threshold.

We first compare the memory and loss of the different estimation strategies. The memory is measured in counters per truly frequent itemset. The average and minimum estimation strategies require consistently fewer counters per frequent itemset than the upper bound estimation strategy (Figures 2(a), 2(c), 3(a), and 3(c)). In terms of loss the inference based estimation strategies are the best. The average estimation strategy has the overall lowest loss, the upper bound strategy the highest (Figures 2(b), 2(d), 3(b), and 3(d)).

To compare our approach with state of the art algorithms, we reimplemented Lossy Counting and Carma. In Figure 4 we compare the counters in memory per truly frequent itemset 4(c) and the loss 4(d) of Carma, Lossy Counting, and Partial Counting. The number of counters per truly frequent itemset is a fair mode of comparison, as it is an implementation independent measure, whereas Megabytes depend more on the internal data representation. Our Partial Counting algorithm clearly outperforms both algorithms in terms of memory and Carma also in loss. Note that the number of counters in memory of Lossy Counting increases as the stream evolves (see Figure 4(c)). The memory required by our Partial Counting algorithm remains low, as the stream evolves. It is not surprising that Lossy Counting performs better in terms of loss for the 10% frequency threshold, as it stores more information yielding typically better results.

We now take a look at the effect of the frequency threshold upon our algorithm. The experiments show that a lower frequency threshold results in a lower loss for our algorithm (Figures 2(b) vs 2(d), 3(b) vs 3(d), and 4(d) vs 4(b)). The effect on memory is the same but less prominent (Figure 2(a) vs 2(c)).



(a) memory for $\Theta = 10\%$ (b) loss for $\Theta = 10\%$

(c) memory for $\Theta = 30\%$ (d) loss for $\Theta = 30\%$

Fig. 4. Memory and loss of CARMA, LOSSY COUNTING, and PARTIAL COUNTING at 10% (top) and 30% (bottom) frequency threshold for the dataset mushroom.

Finally we observe that the loss decreases as the stream evolves (Figures 2(b), 2(d), 4(b) and 4(d)). This trend may be not observed for the short stream in Figure 3.

7 Conclusion

We have presented the PARTIAL COUNTING algorithm for mining frequent itemsets from data streams with a space complexity proportional to the number of frequent itemsets generated from the stream at any point in time. The algorithm starts counting the frequency of an itemset once all of its subsets are estimated to be frequent. The support count of itemsets is estimated by statistical inference.

We have evaluated our algorithm empirically and compared its memory consumption and accuracy with that of CARMA and LOSSY COUNTING. While our algorithm outperforms CARMA in both of these aspects, LOSSY COUNTING turned out to have a better accuracy (measured with Jaccard distance). This is, however, not surprising because LOSSY COUNTING, as our experiments clearly demonstrate, uses significantly more space than Partial Counting. Thus, our algorithm is an alternative to LOSSY COUNTING when memory is limited.

In its current status, PARTIAL COUNTING is outperformed in terms of accuracy by LOSSY COUNTING. As for future work, we would like to reduce this gap. We are going to follow two ideas to achieve this goal. We will experiment with a confidence parameter to reduce the effect of overestimation when we first observe itemsets X and Y together. We believe that we can further improve

our probabilistic inference mechanism based on the observation that an itemset which is not counted, while all of its subsets are counted, must be infrequent.

Another important issue is that, as we will show in the long version of this paper, our algorithm is correct for any infinite data stream. We note that this holds even if we relax the assumption that the underlying distribution is stationary. It is an interesting question that we are investigating, whether concept drift can be handled for finite data streams as well.

Last but not least, we are going to analyse our algorithm for distributed data streams as well. One of the most challenging questions towards this direction is to find a distributed algorithm that generates frequent itemsets of good approximation performance with as few as possible communication.

8 Acknowledgment

This research has been supported by the EU FP7-ICT-2013-11 under grant 619491 (FERARI).

References

1. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Proceedings of the 10th Annual European Symposium on Algorithms. ESA '02, London, UK, UK, Springer-Verlag (2002) 348–360
2. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of the 28th International Conference on Very Large Data Bases. VLDB '02, VLDB Endowment (2002) 346–357
3. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms **55**(1) (2005) 58 – 75
4. Hidber, C.: Online association rule mining. In Delis, A., Faloutsos, C., Ghandeharizadeh, S., eds.: SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA, ACM Press (1999) 145–156
5. Wang, E., Chen, A.: A novel hash-based approach for mining frequent itemsets over data streams requiring less memory space. Data Mining and Knowledge Discovery **19**(1) (2009) 132–172
6. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (1993) 207–216
7. Mannila, H., Toivonen, H., Verkamo, A.I.: Efficient algorithms for discovering association rules. In: Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, July 1994. Technical Report WS-94-03. (1994) 181–192
8. Bache, K., Lichman, M.: UCI machine learning repository (2013)