Alcom-FT Technical Report Series
ALCOMFT-TR-02-6

Sampling Strategies for Finding Frequent Sets *

J. Baixeries
Departament de LSI
Universitat Politècnica de Catalunya
jbaixer@lsi.upc.es

G. Casas-Garriga
Departament de LSI
Universitat Politècnica de Catalunya
gcasas@lsi.upc.es

March 27, 2002

Abstract

There are many deterministic algorithms that perform the task of finding frequent itemsets in a database. All of them depend somehow on the size of the database, and when this size increases, the computational cost of the algorithms becomes higher.

Sampling strategies are a way to overcome this problem. The classical probabilistic approaches for finding frequent sets have been using Chernoff bounds and equivalents to calculate the size of the sample, although the result always overestimates. One of the proposed alternatives is on-line sampling, which adaptatively calculates the size of the sample for each itemset. But this technique has been proved difficult to implement using the current algorithmic schemes for finding frequent itemsets.

In this paper we present different implementations of the on-line scheme using our previous best-first proposal, which meets the requirements needed by online sampling. We compare these proposals with the batch one. Our final approach shows a stable behaviour in several different situations.

1. Introduction

One of the most relevant tasks in Knowledge Discovery in Databases is the discovery of association rules in large masses of data, as it was first formulated in [3]. In this problem, we are given a (large) set of items and a (very large) set of transactions, each of which is a set of items. The "association rule" discovery task identifies the groups of items that most frequently appear along with other groups of items.

There are a great number of data mining problems related to association rules, but here we only restrict ourselves to the subproblem of finding frequent patterns. Moreover we concentrate on algorithms that produce supports of all frequent itemsets, and not only that of maximal ones, since all of them are necessary for all the usual measures of confidence, interest, or conviction in the later phase of constructing association rules.

Even for this basic frequent itemsets problem, there is a rich variety of algorithmic proposals. However, these algorithms have to face different problems, for instance, the size of the database, which can be extremly large and can explode the computational cost of algorithms. Sampling is one of the solutions that has been proposed to overcome this problem since for most data mining applications, approximate answers are also acceptable. Sampling tries to reduce the size of the database, choosing just a fixed number of transactions of it to perform the mining.

In this paper, we will introduce two main current techniques of sampling: *batch sampling* [11] that uses a fixed sample size, and *on-line sampling* [7] which is an adaptative approach. Batch

*This work is supported in part by EU ESPRIT IST-1999-14186 (ALCOM-FT), and matching funds TIC2000-1970-CE, and CIRIT 1997SGR-00366.

sampling is less recommendable because it overestimates the necessary transactions to find out all the frequent itemsets; but on the other hand, on-line sampling turns out to be less applicable with the current algorithmic shemes. Later, we discuss how to adapt current algorithmic proposals for finding frequent sets to these strategies of sampling, and how they can be implemented. Finally, a new model will be proposed to overcome the problems of former proposals.

2. Problem Formulation

The input of a frequent sets algorithm is a collection of *itemsets*, or *transactions* or *records*, each labeled by a unique identifier. Each transaction is a subset of a given fixed set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_N\}$; thus, if we forget about the identifiers, the input would be just a multiset of transactions that we identify as our input *database*. A k-itemset is an itemset consisting of k items.

The *support* of an itemset is the fraction of transactions in the input database that contains the itemset. An itemset is called *frequent* if its support exceeds a given threshold, σ. The frequent sets problem is: given the database and the threshold σ, find all the frequent sets of items.

In most applications, association rules are constructed from the frequent sets and selected or discarded according to criteria related to the supports of the constituting sets. We do not discuss here how to combine frequent sets into association rules, (see [4], [6], or references in [9]); but we do point out that, in order to compute association rules, one needs not only the list of maximal frequent itemsets but also the supports of all frequent itemsets, whether maximal or not.

2.1. Algorithmic Strategies for Finding Frequent Sets

Many algorithmic strategies have been designed to perform the task of finding frequent itemsets in a database. Broadly speaking, these strategies can be classified into two main ways, according to how they address the problem:

- Traversing the lattice of itemsets in search of either all frequent sets or at least all the maximal ones.
- Computing projections of the input database in order to reduce it to a smaller data structure.

Algorithms belonging to the first group can be classified according to the way they explore the induced lattice of the database, which will be done processing the records in that database. This lattice is the representation of all the possible frequent sets that can be formed from $\mathcal{I}$. As analized in [9], the lattice-traversal algorithms can be classified into three high-level frameworks: either they focus on counting the support in a breadth-first fashion, or on finding frequent itemsets by intersecting tid lists of transactions using a depth-first approach, or finally, [5] proposes a best-first strategy, where the exploration deepens early on those itemsets that are easy to certify as frequent.

We will focus in this paper on the algorithms that follow this first framework, and, thus, our main aim will be reducing as much as possible the number of transactions needed to achieve our task.

When it comes to algorithms belonging to the second group, like FPGrowth [8] or TreeProjection algorithm of [2], we will later argue why current sampling strategies are difficult to be applied in this group of algorithms.

2.1.1 Breadth-first Algorithms

Algorithms included in this category traverse the induced lattice bottom-up. These algorithms' basic structure consists in finding how many itemsets of size n are frequent, and, at a given point, using them to generate candidates of size $n+1$. Then, the same scheme is applied over and over until no more candidates are generated.

The best known breadth-first algorithm is Apriori [4]. It devotes one pass through the database to each group fo itemsets of size k. Along each pass, it determines the support of all the candidate itemsets of that size. Then, a candidate generation is performed using the antimonocity property of the frequent sets in order to determine the potential candidates of size $k+1$, and so on.

There are a number of variations of this algorithm, for instance in terms of the data structure where current frequent itemsets are stored. A special approach that can be included in this category (also in the following) consists in constructing, for each itemset, a list of all the identifiers of the transactions where the frequent itemsets appear. See the survey [9] for a more accurate description.

A variation of Apriori is the Dynamic Itemset Counting (DIC) algorithm [6], which partially retains the breadth-first strategy but tries to jump one level up every M transactions. This technique results in less visited records than Apriori because of the earlier incorporation of new candidates into the counting.

2.1.2 Depth-first Algorithms

These algorithms explore the induced lattice in a form close to a depth-first search. It is easy to see that the plain depth-first approach on the itemsets is not that effective. Instead, the itemsets are organized in classes, and these classes are in turn organized in a tree-like form that is, in a sense, a sort of (quocient of a) subgraph of the itemset lattice, and it is this tree which is traversed depth-first. Some algorithms of this sort were proposed in [12], clear examples appear in [8], and quite a complete description is given in [1].

It has been argued that counting supports is inefficient in depth-first search, and indeed most depth-first proposals construct instead, for each frequent itemset, the whole list of transactions in which it appears, that is a tid list. Then, the algorithms walk up through the lattice by intersecting them, and complement the procedure with various heuristics like fast intersections [12].

It turns out that the counting approach is experimentally found better than the list intersection approach only for the processing of pretty small itemsets, at least on prototypical market basket data. An effective hybrid algorithm that starts like Apriori and later on switches to computing list intersections (and thus starts with breadth-first search and later on switches to depth-first search) has been proposed and studied in [10].

2.1.3 A Best-First Strategy

This proposal in [5] is based on the idea of advancing as much as possible the starting point for counting the support of a candidate itemset so that frequent itemsets detected early will lead sooner to the exploration of larger itemsets. To count the support of a candidate, one needs to fully scan the database, but the order is irrelevant; thus, if we see the input database as "cyclic", we can start counting the support of an itemset at any arbitrary transaction, wrap up to the beginning upon reaching the end, and stop one line earlier than the start. The use of this property leads to the Dynamic Itemset Counting, which allows to generate new candidates at specific moments of the database scan: one profits from the fact that we may know that an itemset is already frequent before finishing counting its support. Thus: the concept of "best" is precisely defined by how early the algorithm can realize that an itemset is, in fact, potentially frequent.

The strategy from [5] avoids the time-consuming process of generating a new round of candidates from the supports found up to the point; instead, we can keep track of how many immediate subitemsets of a given itemset have already been proved frequent. At the moment of receiving the last of such confirmation, the itemset is declared potentially frequent, and its support is started to be counted right there without delay. The earlier we know that all subitemsets of an itemset are frequent, the earlier we start counting its support, since it will not be pruned by the antimonocity property.

Along the exploration, the algorithm following this strategy, maintains two collections of itemsets:

- The **candidate itemsets**, whose support is being counted in the database.
- The **borderline itemsets**, immediate supersets of some candidate itemset that are known to be frequent, and that might become frequent themselves.

Whenever each candidate is proved frequent, the algorithm records this fact appropriately, in such a way that, for borderline itemsets having all their subitemsets frequent, we know it literally as soon as possible, so as to start counting them immediately. All these incorporations of new candidates into the counting is not time-consuming. For more details on the structures used see [5].

Of course, the algorithm must keep track of the transaction where counting started for each candidate in order to know when the corresponding database pass is complete; then the support count is correct for that candidate.

The most interesting point of this strategy is that it seems to be flexible enough (from a structural point of view) to be adapted in a large variety of frameworks, for instance, sampling and adaptative sampling, as we will later see.

3. Current Sampling Strategies

A natural way of reducing the computational costs of many data mining problems is sampling, because it may reduce the number of visited transactions. In this section we present two current sampling strategies: batch and on-line.

A classical sampling strategy in the case of frequent sets is using a *batch sampling* method [11]. In this approach one samples randomly a portion of the database, of a predetermined size, and uses a smaller threshold σ' to compute frequent sets on the sample. The different threshold σ' reduces the risk of missing frequent sets due to the sampling process. Thus, to label a single candidate set X as frequent or non-frequent using a batch sample, an algorithm must take into account the support of X on the fixed sample. In particular, let τ be the fixed sample size, then the classification criterion of *any* of such algorithms after having read a new transaction is the following:

Update support of X
if $\frac{support(X)}{\tau} \geq \sigma'$ **then**
Label X as frequent
else if X has been counted in τ records **and** it is not frequent **then**
Label X as non-frequent

Although the algorithm can label the itemset X as frequent before having processed all the τ transactions in the sample, it must keep on reading the rest of the transactions in order to update an approximation to the real support of X, so, at last, X will have visited τ transactions.

The sample size τ can be computed using concentration inequalities, usually Chernoff or Hoeffding bounds. However, it is important to highligth that because the sample size is calculated a priori, it must be big enough to work well in all the future situations. This means that the bounds used to compute the sample size are nonadaptive, fully oblivious to the supports found, and likely to overestimate the necessary sample sizes.

An alternative approach is *on-line sampling* [7]. Using it, the support of each candidate is evaluated on a growing sample, obtained by picking transactions one after the other randomly; the method provides a probabilistic estimate of the actual support on the whole database, based both on the number of transactions seen so far and the support found on them. Whereas for some cases this method requires the worst-case sample size given by Chernoff bounds, for clear-cut cases only a small fraction of that sample size is actually needed. Indeed, the situation that forces the strategy to require a sample size as big as the one provided by Chernoff bounds is for those itemsets whose real support is clearly close to σ'.

Essentially, the intuition of the method is that one keeps sampling random transactions from the database and updating the frequencies of the current candidates as though it was the next one read; a formula derived in [7] allows us to decide whether the current frequency estimate for each candidate is high and faithful enough to label it as frequent. To be more precise, let X be a candidate we want to classify (that is, label it as frequent or label it as non-frequent), and let S_t the processed sample up to the moment containing t transactions, then the classification criterion used by on-line sampling is, schematically, the following:

Update support of X
if $\frac{support(X)}{S_t} > \sigma' + \epsilon_t$ **then**
    Label X as frequent
else if $\frac{support(X)}{S_t} < \sigma' - \epsilon_t$ **then**
    Label X as non-frequent

Whereas in batch sampling all the τ transactions were compulsoryly processed for a given X, here once the set X has been classified, there is no need to keep on the mining, since the support of X up to the moment will serve as a good aproximate of its real support.

For those situations in which X cannot be classified, the algorithm must keep on reading new transactions until it classifies X, or until it reaches the τ transactions bound of the batch sample.

The exact value for ϵ_t can be found in [7]. It is to be noted that ϵ_t is updated after each transaction, so, the formula allows each candidate to be classified in a adaptative way while reading new transactions. It is proved in [7] that the formula is accurate with exponentially decreasing probability of error.

Any sample method must take into account parametres of the task, an accuracy parameter, and a confidence parameter to determine appropiate number of transactions needed to solve the desired problem. Indeed, when using any sampling strategy only the estimated support of the sets will be available for the true supports in order to compute association rules. This problem can be overcome, if necessary, by a new pass on the database, counting actual supports and checking them with respect to σ, as explained in [11].

4. Using Sampling Strategies for Finding Frequent Sets

Theoretically, the on-line sampling approach seems to act better than the batch approach, or at least it requires fewer processed transactions to find the support of a single itemset (see [7]). However, the problem appears when one needs to find the support of all the frequent itemsets in the database, because other considerations must be taken into account. So, whereas any algorithm for finding frequent sets can be applied over a fixed sample size using a user-specified σ', it is not straightforward how to apply a good algorithmic strategy that can take benefit of the adaptativeness of the on-line sampling.

4.1. Algorithms over On-Line Sampling

Breadth-first algorithms such as Apriori or DIC, need to know in advance a halting point where to start generating new candidates. In case of Apriori this point is the end of the database, and in case of DIC the stop occurs after a certain number of transactions given by the user. With this criterion, neither Apriori nor DIC can benefit from the on-the-fly classification of the strategy, since the halting point is not adaptative during the run and depends on the user. This means that, although the on-line classification criterion can be very fast classifying the candidates, both Apriori or DIC would have to wait for the halting point where to start incorporating new candidates. This behaviour would be the same in the batch approach and the on-line classification would not be so advantageus as it was to be expected.

On the other hand, it is difficult to apply depth-first algorithms over on-line sampling. In fact, it makes no sense to intersect the tid lists of different candidates since their processed transactions may no be the same, and the intersection might be even empty. These schemes can be only applied over a fixed portion of the database.

Different algorithmic approaches like FPGrowth are not truly compatible with on-line sampling. That algorithm is *not* a lattice-traversal algorithm; instead, it distills a different data structure from the input database that has the necessary information for computing frequent sets. The on-line sampling adaptativeness does not serve to construct this structure.

On the whole, we can say that these presented drawbacks when applying the classical algorithms have made the on-line sampling method an strategy not easily applied in the task of finding frequent sets. For overcoming this problem we propose to use the best-first strategy [5], which has the property of incorporating new candidates on the fly.

In fact, it is quite easy to combine on-line sampling with best-first search; in particular, the algorithm in [5] is particularly appropriate to be combined with this sampling strategy. This is because of the fact that borderline itemsets can become candidates at any moment during the run, which follows the adaptiveness of on-line sampling: in essence, the only difference is that we do not need to wait until the whole database (or sample) has been seen to finish counting a candidate: the formula in [7] not only gives the halting criterion but also the labeling criterion for an itemset. We will call *On-LineBF* this application of the best-first algorithmic strategy over on-line sampling.

To be more precise with the proposal, at any moment during the run, On-LineBF keeps counting the support of the current candidates in random transactions. At the same time, the on-line sampling criterion, using the formula in [7], can decide to label a candidate as frequent or as non-frequent. As soon as a borderline itemset has all its subitemsets labeled as frequent, it is incorporated into the counting. This matches exactly the adaptativeness flavour of the on-line sampling, and it has the advantage that the incorporation of new candidates is made without waste of time (see [5] for more details). A more precise formal description of the algorithm *On-LineBf* is the following pseudocode, where $\mathcal{C}$ is the set of current candidates and $\mathcal{B}$ is the set of current borderline itemsets.

τ := Chernoff bound
<u>**while**</u> $\mathcal{C} \neq \emptyset$ <u>**do**</u>
    Read a random transaction
    <u>**for all**</u> $X \in \mathcal{C}$ <u>**do**</u>
        Update support of X
        <u>**if**</u> X has processed τ transactions and has not still been classified <u>**then**</u>
            use *batch_classification_criterion*
        <u>**else**</u>
            use *on-line_classification_criterion*
        <u>**end**</u>
        <u>**if**</u> any Y, where $Y \in \mathcal{B}$ and $X \subset Y$, has every subset frequent <u>**then**</u>
            $\mathcal{C} := \mathcal{C} \cup \{Y\}$
        <u>**end**</u>
    <u>**end**</u>
<u>**end**</u>

It is worth to note that On-LineBf only uses the batch classification criterion when the candidate has not been classified after reading all the τ transactions. Moreover, it is important to describe the procedures *batch_classification_criterion* and *batch_classification_criterion*, which use the criteria described previously by sampling strategies, but adding some steps from the best-first algorithm:

batch_classification_criterion

if $\frac{support(X)}{\tau} \geq \sigma'$ **then**
  Label X as frequent
  Notify the frequency of X to all Y, where $X \subset Y$ and $Y \in \mathcal{B}$
else if X has already processed τ transactions **then**
  $\mathcal{C} := \mathcal{C} \setminus \{X\}$
  if $\frac{support(X)}{\tau} \leq \sigma'$ **then**
    Label X as non-frequent
  end
end

Note that when a candidate X is declared frequent using this batch classification system, it is not eliminated from $\mathcal{C}$.

on-line_classification_criterion

if $\frac{support(X)}{S_t} > \sigma' + \epsilon_t$ **then**
  Label as X frequent
  $\mathcal{C} := \mathcal{C} \setminus \{X\}$
  Notify the frequency of X to all Y, where $X \subset Y$ and $Y \in \mathcal{B}$
else if $\frac{support(X)}{S_t} < \sigma' - \epsilon_t$ **then**
  Label X as non-frequent
  $\mathcal{C} := \mathcal{C} \setminus \{X\}$
end

In fact, the only syntactical difference between On-LineBF and the deterministic best-first algorithm of [5] is the classification criterion for every candidate: in On-LineBF it is given by the formula in [7], while in the deterministic fashion depends on the number of occurrences of that cadidate over the whole database.

4.2. Algorithms over Batch Sampling

In [11] the Apriori algorithm is applied over a fixed sample size; but in fact, any algorithmic strategy can be applied over a portion of the database without loss of generality. However, as we are interested in comparing the above proposed algorithm against batch sampling, it is natural to use the best-first strategy over a predetermined sample size as a competitor. In [5] we show how this strategy processes less transactions than breadth-first algorithms when using the whole database, and this behaviour is also to be expected when using a fixed sample size.

We will call *BatchBF* the algorithm resulting of the application of the best-first strategy over a batch sample. A more formal description of the BatchBF algorithm is the following pseudocode:

$\tau :=$ Chernoff bound
$S :=$ Extract τ transactions from the database
while $\mathcal{C} \neq \emptyset$ **do**

```
        Read next transaction from S
        for all  X ∈ C do
            Update support of X
            use batch_classification_criterion
            if  any Y, where Y ∈ B and X ⊂ Y, has every subset frequent  then
                C := C ∪ {Y}
            end
        end
    end
```

In this pseudocode, the *batch_classification_criterion* is the same described formerly.

Thus, a batch sampling method combined with best-first approach turns out to be the best choice for a fair comparison; so we have developed both versions of best-first search based on sampling, one using batch sampling and one using online sampling.

4.3. Experiments

In this section, we will comment the results of the experiments performed in order to compare our implementation for batch sampling and on-line sampling using for both the best-first strategy. Our main aim was to force the classification criteria of both algorithms, so that we could see different behaviours according to the different situations. It is worth to note that both BatchBF and On-LineBF are constructed using the same algorithmic base, so, the comparison of both implementations will be done in terms of number of visited records.

We have performed several experiments using a synthetic database generated by the Almaden generator[1], and were run in a Pentium at 300 Mh with 256 Mb of main memory under Linux. The database contained one milion records, and the average record size was 10 items of of a choice of 100.

We ran our experiments using a one-milion-record database, varying the support from 5% to 0.2%, and comparing algorithms in terms of processed transactions. The reduced support σ' represents in all experiments a 10% reduction of the support σ.

		BatchBF		**On-LineBF**	
σ	σ'	**Records**	**Number of frequents**	**Records**	**Number of frequents**
5%	4.5%	245508	222	113713	223
3%	2.7%	746064	571	308744	570
2%	1.8%	1667139	1188	691400	1186
1%	0.9%	6810301	3910	2735429	3910

Table 1: Processing fewer records by On-LineBF: synthetic data

In table 1 we can see the total number of records visited by each algorithm, as well as the number of frequent itemsets found for each minimum support and algorithm. Clearly, the on-line strategy performs always better than the batch strategy, and the number of frequent itemsets found is likely the same. On-line sampling outperforms batch in a factor of $40-45\%$, regardless of the minimum support and error rate. Moreover, the probabilistic approaches shown below do not depend on the size of the database, so it is to be expected that this behavior would be the same regardless of the size of the database.

It is worth to mention that both BatchBF and On-LineBF perform always better than any deterministic algorithm. For instance, when $\sigma = 2\%$ the larger candidate is a 10-itemset, which means that an algorithm like Apriori would visit 10 million records (since the size of the database is of 1 million records).

[1] www.almaden.ibm.com/cs/quest/syndata.html

The next step was to perform tests using real databases. We used a sample of the USA census from PUMS[2] consisting of 3000-transaction database of 80 possible items. It is worth to note that the database is so small that it could be argued that it was not necessary to use sampling. Obviously, when the support σ decreases, the number of transactions needed by a sampling algorithm will increase too much, and aplying a sampling strategy will be worthless. But if the support is not too low, applying BatchBF or On-LineBF will still be a benefit, since they will process fewer transactions than a deterministic algorithm. This is why we only perform experiments up to a support of 30%, because for lower supports, the sample oversizes the number of transactions needed by a deterministic algorithm.

		BatchBF	**On-LineBF**
σ	σ'	**Records**	**Records**
50%	45%	4644	5047
40%	36%	5308	5707
30%	27%	6466	7415

Table 2: Processing fewer records by batch sampling: real data

As table 2 shows, in this case batch sampling did perform better that on-line sampling. When it comes to the number of frequent itemsets, both On-LineBF and BatchBF performed similarly and discovered almost the same group of frequent itemsets.

Thus, from the previous experiments we have seen that there are situations where BatchBF is better than On-LineBF, and viceversa. In fact the reason for these different behaviours is quite simple, and it can be found in how fast an algorithm (BatchBF or On-LineBF) is able to classify their candidates.

To be more precise with this important fact, we have discused in the best-first strategy that the sooner the candidates are declared frequent, the sooner the best-first scheme can be used, and so, the sooner borderline itemsets (that is potential candidates) are incorporated into the counting. This results in processing fewer transactions at the end of the whole process of mining. We have argued in [5] that the fast (early) incorporation of new candidates makes an algorithm processing fewer transactions, and in fact, this is one of the reasons why the deterministic algorithm in [5] visits fewer transactions than Apriori or DIC.

What happens in the example of real databases is that batch classification criterion is faster because it can label a candidate frequent before it has been completely counted in all the τ transactions (following the best-first strategy). This does not means that this candidate is eliminated from the set of candidates $\mathcal{C}$, since we are in a batch approach and all the τ transactions will be needed to count an approximation of its real support; but at least, best-first strategy allows BatchBF to be clever enough to label candidates earlier. This will benefit the immediate incorporation of new candidates into $\mathcal{C}$. In the same situation, On-LineBF does not use the best-first intuitions to label unless it has visited all the τ transactions, and in general, tries to classify candidates following the formula in [7]. This may cause that in some cases more transactions than BatchBF are needed to realize that a candidate is frequent. Thus, in this example On-LineBF is slower in the classification.

In the example of synthetic databases happens quite the reverse. We are in front of a kind of data which causes BatchBF to need more transactions to label a candidate, whereas On-LineBF and the formula in [7] is clever in these situations. All this helps On-LineBF in the faster incorporation of new candidates into $\mathcal{C}$, and consequently the visited number of transactions is smaller.

Thus, it is clear from the experiments that we have the possibility that the batch method would declare frequent an itemset sooner than the on-line method and viceversa. In the next section we will see how to overcome these problems with a new algorithmic proposal able to always outperform

[2] www.ipums.umn.edu/usa/intro.html

BatchBF and On-LineBF.

5. New Algorithmic proposal over On-Line Sampling

As we have seen, we are interested in reducing as much as possible the number of processed transactions since the resulting algorithm will be less time consuming. In the last experiments, we were expecting that the on-line sampling criterion would classify a candidate faster (using less transactions) than the batch criterion. Nevertheless, the results show that not always a best-first strategy over on-line sampling is a better choice than the same strategy over a batch sampling.

We have argued in the previous section that the reason that explains this behaviour is that not only On-LineBF can classify an itemset before it has counted τ transactions, but also BatchBF can do it as well. So, there is no clear reason why any of both algorithms should behave better than the other in all circunstances.

It is then obvious that using one of boths algorithms does not guarantee that it will result in processing fewer records than the other when mining a database. Our need, in order to avoid this problem, is to construct an algorithm that classifies an itemset as fast as the fastest of the previous algorithms. The point is that for certain itemsets, batch criterion will be a faster classifier, and that for others, on-line criterion would be the right choice. In fact, we will need to have both criterions available in order to decide for each itemset, the best one. Apart from this, we would need, as well, a sort of oracle that, given a candidate, is able to answer which criterion to use: whether batch or on-line.

Thus, in this new clever algorithm, before trying to classify a candidate, it should first know which of both criterions (batch or online) would classify it faster, given an itemset candidate, its current support and the number of records already visited for that itemset. According to the answer, the algorithm would then try to classify this itemset according to this best criterion. The same question would be set to the oracle until the itemset is classified. We call this proposal *On-LineBBF*.

Obviously, this oracle is not available, but it can be simulated using the appropriate algorithmic scheme. Formally, the pseudocode for On-LineBBF is the following.

```
τ := Chernoff bound
while    C ≠ ∅  do
    Read a random transaction
    for all  X ∈ C do
        Update support of X
        use batch_classification_criterion
        if  X is not still classified  then
            use on-line_classification_criterion
        end
        if  any Y, where Y ∈ B and X ⊂ Y, has every subset frequent  then
            C := C ∪ {Y}
        end
    end
end
```

It is worth to note that the pseudocode for On-LineBBF is really similar to On-LineBF, but with the main difference that the guard before of the use of *batch_classification_criterion* has disappeared.

5.1. Experiments and Conclusions

With the aim of comparing the new proposal with BatchBF and On-LineBF the same experiments are performed now with On-LineBBF. The results on synthetic data can be seen in table

3, and the results on real data are in table 4.

We can observe that On-LineBBF visits around 2% fewer transactions than On-LineBF in synthetic data, and of course, fewer transactions than BatchBF. Moreover, the number of final frequent itemsets found is clearly similar to On-LineBF and BatchBF, which shows that the classification error of these algorithms is almost the same.

		On-LineBBF	
σ	σ'	**Records**	**Number of frequents**
5%	4.5%	111645	223
3%	2.7%	307478	570
2%	1.8%	677521	1187
1%	0.9%	2697073	3910

Table 3: Processing fewer transactions by On-LineBBF: synthetic data

For the real database results, On-LineBBF performs also better than On-LineBF and BatchBF. Indeed, in most cases On-LineBBF and BatchBF visit almost the same number of records, which is not a drawback since our main aim was constructing an algorithm able to be always the best on any situation.

		On-LineBBF
σ	σ'	**Records**
50%	45%	4644
40%	36%	5135
30%	27%	6407

Table 4: Processing fewer transactions by On-LineBBF: real data

In this section we have shown how the last proposal On-LineBBF performs better than On-LineBF and BatchBF in all tests. The algorithmic scheme of On-LineBBF guarantees that this behaviour will persist regardless of the kind of data in differents supports. The reason is that this scheme is a generalization of the previous approaches. What is more, the error commited in the classification is almost the same in the three proposals, which leads to think that On-LineBBF is the best option of the three.

References

[1] J.M. Adamo. *Data Mining for Association Rules and Sequential Patterns.* Springer-Verlag 2001.

[2] R. Agrawal, C. Aggarwal, and V. Prasad. "A Tree Projection Algorithm for Generation of Frequent Item Sets". *Journal of Parallel and Distributed Computing*, 350–371. 2000.

[3] R. Agrawal, T. Imielinski, and A. Swami. "Mining Association Rules Between Sets of Items in Large Databases". *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, 207–216. 1993.

[4] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. "Fast Discovery of Association Rules". In: *Advances in Knowledge Discovery and Data Mining*, 307–328. AAAI Press, 1996.

[5] J.Baixeries, G.Casas-Garriga and J.L.Balcázar. "A Best First Strategy for Finding Frequent Sets". *Extraction et gestion des connaissances* (EGC'2002), 100–106. 2002.

[6] S. Brin, R. Motwani, J. Ullman, and S. Tsur. "Dynamic Itemset Counting and Implication Rules for Market Basket Data". *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, 255–264. ACM Press, 1997.

[7] C. Domingo, R. Gavaldà, and O. Watanabe. "Adaptive Sampling Methods For Scaling Up Knowledge Discovery Algorithms".*Data Mining and Knowledge Discovery*, 131–152. 2002.

[8] J. Han, J. Pei, and Y. Yin. "Mining Frequent Patterns Without Candidate Generation". *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, 2000.

[9] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. "Algorithms for Association Rule Mining – A General Survey and Comparison". *ACM SIGKDD Explorations.* Vol 2, Issue 2, 58–64. 2000.

[10] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. "Mining Association Rules: Deriving a Superior Algorithm by Analysing Today's Approaches". *Proc. of the 4th. European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'2000)*, 159–168, 2000.

[11] H. Toivonen. "Sampling Large Databases for Association Rules". *Proceedings of the 22nd Int'l Conf. on Very Large Databases*, 134–145. 1996.

[12] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. "New Algorithms For Fast Discovery of Association Rules". *Proc. of the 3rd. Int'l Conf. on KDD and Data Mining* (KDD'97). 1997.