Prog Artif Intell (2012) 1:45–55
DOI 10.1007/s13748-011-0002-6

REVIEW

A survey on learning from data streams: current and future trends

João Gama

Received: 2 February 2011 / Accepted: 1 July 2011 / Published online: 13 January 2012

Abstract Nowadays, there are applications in which the data are modeled best not as persistent tables, but rather as transient data streams. In this article, we discuss the limitations of current machine learning and data mining algorithms. We discuss the fundamental issues in learning in dynamic environments like continuously maintain learning models that evolve over time, learning and forgetting, concept drift and change detection. Data streams produce a huge amount of data that introduce new constraints in the design of learning algorithms: limited computational resources in terms of memory, cpu power, and communication bandwidth. We present some illustrative algorithms, designed to taking these constrains into account, for decision-tree learning, hierarchical clustering and frequent pattern mining. We identify the main issues and current challenges that emerge in learning from data streams that open research lines for further developments.

Keywords Data mining · Machine learning · Learning from data streams

1 Introduction

Informally speaking, the goal of machine learning is to build a computational model from the past experience of what has been observed. Machine learning studies automatic methods for acquisition of domain knowledge with the goal of improving system's performance as the result of experience. In the past two decades, machine learning research and practice has focused on batch learning usually with small data sets. In batch learning, the whole training data are available to the algorithm that outputs a decision model after processing the data eventually (or most of the times) multiple times. The rationale behind this practice is that examples are generated at random accordingly to some stationary probability distribution. Most learners use a greedy, hill-climbing search in the space of models. They are prone to overfitting, local maxima's, etc. Data are scarce and statistic estimates have high variance. A paradigmatic example is the TDIT algorithm to learn decision trees [48]. As the tree grows, less and fewer examples are available to compute the sufficient statistics, variance increase leading to model instability. Moreover, the growing process re-uses the same data, exacerbating the overfitting problem. Regularization and pruning mechanisms are mandatory.

The developments of information and communication technologies dramatically change the data collection and processing methods. What distinguish current data sets from earlier ones are *automatic data feeds*. We do not just have people entering information into a computer. We have computers entering data into each other [46]. Moreover, advances in miniaturization and sensor technology lead to sensor networks, collecting high-detailed spatio-temporal data about the environment.

An illustrative application is the problem of mining data produced by sensors distributed all around electrical-power distribution networks. These sensors produce streams of data at high-speed. From a data mining perspective, this problem is characterized by a large number of variables (sensors), producing a continuous flow of data, in a dynamic non-stationary environment. Companies analyze these data streams and make decisions for several problems. They are interested in identifying critical points in load evolution, e.g., peaks on the demand. These aspects are related to anomaly

J. Gama (✉)
LIAAD-INESC-Porto LA, and FEP-University of Porto,
R. de Ceuta 118-6, 4050 Porto, Portugal
e-mail: jgama@fep.up.pt

detection, extreme values, failures prediction, outliers and abnormal activities detection. Other problems are related to change detection in the behavior (correlation) of sensors. Cluster analysis can be used for the identification of groups of high-correlated sensors, corresponding to common behaviors or profiles (e.g., urban, rural, industrial, etc.). Decisions to buy or sell energy are based on the predictions on the value measured by each sensor for different time horizons. Data mining, in this context, require continuous processing of the incoming data monitoring trends, and detecting changes. Traditional one-shot systems, memory based, trained from fixed training sets and generating static models are not prepared to process the high-detailed data available, are neither able to continuously maintain a predictive model consistent with the actual state of the nature, nor react quickly to changes.

In this article, we discuss the issues and challenges on learning from data streams discussing limitations of the current learning systems and pointing out possible research lines for next generation data mining systems. The paper is organized as follows. The next section identifies the main characteristics of streaming algorithms and present illustrative sublinear algorithms to obtain approximate answer in querying high-speed streams. The Sect. 3 presents the illustrative streaming algorithms for three learning tasks: decision trees, clustering and frequent pattern mining. Section 4 identifies challenges and open issues in the current research in stream mining. Last section concludes the paper identifying the lessons learned.

2 Machine learning and data streams

Machine learning extracts knowledge (models, patterns) from data and the nature of data is changing. Nowadays, we have technology to continuously capture digital data. Data are generated and collected at high speed, meaning that the data arrival rate is high relative to the computational power. In these applications, data are modeled best not as persistent tables, but rather as transient data streams [6] (Table 1).

Table 1 Summary of the main differences between the standard database processing and data stream processing

	Databases	Data streams
Data access	Random	Sequential
Number of passes	Multiple	Single
Processing time	Unlimited	Restricted
Available memory	Unlimited	Fixed
Result	Accurate	Approximate
Distributed	No	Yes

2.1 Streaming algorithms

In the streaming model, the input elements $a_1, a_2, \ldots, a_j, \ldots$ arrive sequentially, item by item [46]. We can distinguish between two different models:

1. Insert only model: once an element a_i is seen, it cannot be changed;
2. Insert-delete model: elements a_i can be deleted or updated.

From the view point of a Data Streams Management Systems (DSMS) [18], several research issues emerge that require approximate query processing techniques to evaluate continuous queries that require unbounded amount of memory [8]. A relevant issue is the definition of the semantics (and implementation) of blocking operators (operators that only return an output tuple after processing all input tuples, like join, aggregation and sorting) in the presence of unending streams.

Algorithms that process data streams deliver approximate solutions, providing a fast answer using a few memory resources; they relax the requirement of an exact answer to an approximate answer within a small error range with high probability. In general, as the range of the error decreases, the space of computational resources goes up.

Illustrative example. Suppose the problem of counting the number of distinct pairs of IP addresses from the network traffic that crosses a server which is a trivial problem, if we do not consider restrictions in space. The number of distinct pairs of IP's can be very large, and an exact answer is not mandatory.

Hash functions are a powerful tool in stream processing. They are used to project huge domains into lower space dimensions. One of the earlier results is the Hash (aka FM) sketches for distinct-value counting in one pass while using only a small amount of space [26]. Suppose the existence of a hash function $h(x)$ that maps incoming values $x \in [0, \ldots, N-1]$ uniformly across $[0, \ldots, L-1]$, where $L = O(\log N)$. Let $\text{lsb}(y)$ denote the position of the least-significant 1 bit in the binary representation of y. A value x is mapped to $\text{lsb}(h(x))$. The algorithm maintains a bitmap vector of L bits, initialized to zero. For each incoming value x, set the $\text{lsb}(h(x))$ bit of L to 1. At each time-stamp t, let R denote the position of rightmost zero in the bitmap. Flajolet and Martin [26] prove that R is an indicator of $\log(d)$, where d denotes the number of distinct values in the stream.

In some applications, mostly database oriented, an approximate answer should be within an admissible error margin. DSMS developed a set of techniques that store compact stream summaries enough to approximately solve queries. All these approaches require a trade-off between accuracy and the amount of memory used to store the summaries, with

an additional constrain of small time to process data items [2,46]. The most common problems end up to compute quantiles, frequent itemsets, and to store frequent counts along with error bounds on their true frequency. The techniques developed are used in very high dimensions both in the number of examples and in the cardinality of the variables.

2.2 Approximation and randomization

Many fundamental questions, like counting, require space linear in the input to obtain exact answers. Within data stream framework, *approximation* techniques, that is, answers that are correct within some small fraction ϵ of error; and *randomization* [45], that allow a small probability δ of the failure, are used to obtain answers with that of the probability $1-\delta$ are in an interval of radius ϵ. Algorithms that use both approximation and randomization are referred to as (ϵ, δ) approximations. The base idea consists of mapping a very large input space to a small synopsis of size $O(\frac{1}{\epsilon^2} log(\frac{1}{\delta}))$.

Approximation and randomization techniques has been used to solve problems like measuring the entropy of a stream [17], association rule mining [4], frequent items [44], k-means clustering for distributed data streams using only local information [21], etc.

2.3 Time windows

Most of the time, we are not interested in computing statistics over all the past, but only over the *recent* past. The assumption behind all these models is that the most recent information is more relevant than the historical data. The simplest situation uses *sliding windows* of fixed size. These types of windows are similar to *first in, first out* data structures. Whenever an element j is observed and inserted into the window, another element $j-w$, where w represents the window size, is forgotten.

Several window models have been presented in the literature. Babcock et al. [7] define two basic types of sliding windows:

- **Sequence based.** The size of the window is defined in terms of the number of observations. Two different models are *sliding windows* of fixed size w and *landmark windows*, where the window grows from a specific point in time (the landmark);
- **Time-stamp based.** The size of the window is defined in terms of *duration*. A time-stamp window of size t consists of all elements whose time-stamp is within a time interval t of the current time period.

Monitoring, analyzing and extracting knowledge from high-speed streams might explore multiple levels of granularity, where the recent history is analyzed at fine levels of granularity and the need of precision decreases with the age of the data. As a consequence, the most recent data can be stored at the finest granularity, while more distant data at coarser granularity. This is called the *tilted* time window model. It might be implemented using exponential histograms [24].

Sequence-based windows is a general technique to deal with changes in the process that generates data. A reference algorithm is the `AdWin-ADaptive sliding WINdow` presented by Bifet and Gavaldà [10]. `AdWin` keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis *there has been no change in the average value inside the window*. More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: first, that change is reliably declared whenever the window shrinks; and second, that at any time the average over the existing window can be reliably taken as an estimate of the current average in the stream (barring a very small or very recent change that is still not statistically visible). `AdWin` is parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, a property inherent to all algorithms dealing with random processes. `AdWin` does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique. This means that it keeps a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item.

2.4 Sampling

Sampling is a common practice for selecting a subset of data to be analyzed. Instead of dealing with an entire data stream, we select instances at periodic intervals. Sampling is used to compute statistics (expected values) of the stream. While sampling methods reduce the amount of data to process, and, by consequence, the computational costs, they can also be a source of errors, namely in monitoring applications that require to detect anomalies or extreme values.

The main problem is to obtain a *representative* sample, i.e., a subset of data that has approximately the same properties of the original data. In statistics, most techniques require to know the length of the stream. For data streams, we need to modify these techniques. The simplest form of sampling is *random sampling*, where each element has equal probability of being selected [1]. The *reservoir sampling* technique [53] is the classic algorithm to maintain an online random sample. The base idea consists of maintaining a sample of size k, called the reservoir. As the stream flows, every new element has a probability k/n, where n is the number

of elements seen so far, of replacing an old element in the reservoir.

A similar technique, load shedding, drops sequences in the stream, when bursts cause bottlenecks in the processing capabilities. Tatbul et al. [51] discuss load shedding techniques in querying high-speed data streams.

2.5 Synopsis, sketches and summaries

Synopsis is compact data structures that summarize data for further querying. Several methods have been used, including: wavelets [35], exponential histograms [24], frequency moments [5], etc. Data sketching via random projections is a tool for dimensionality reduction. Sketching uses random projections of data points with dimension d to a space of a subset of dimensions. It has been used for moment estimation [5], computing L-norms [46] and dot product of streams [1].

Cormode and Muthukrishnan [20] presents a data stream summary, the so-called *count-min* sketch, used for (ϵ, δ) approximations to solve *point queries*, *range queries*, and *inner product queries*. Consider an implicit vector **a** of dimension n that is incrementally updated over time. At each moment, the element a_i represents the counter associated with element i. A point-query is to estimate the value of an entry in the vector **a**. The count-min sketch data structure, with parameters (ϵ, δ), is an array of $w \times d$ in size, where $d = \log(1/\delta)$, and $w = 2/\epsilon$. For each incoming value of the stream, the algorithm use d hash functions to map entries to $[1, \ldots, w]$. The counters in each row are incremented to reflect the update. From this data structure, we can estimate at any time, the number of occurrences of any item j by taking $\min_d \text{CM}[d, h_d(j)]$. Since the space used by the sketch CM is typically much smaller than that required to represent the vector **a** exactly, there is necessarily some approximation in the estimate. The estimate $\hat{a_j}$, has the following guarantees: $a_j \leq \hat{a_j}$, and, with probability at least $1-\delta$, $\hat{a_i} \leq a_i + \epsilon ||a||_1$. The error of the estimate is at most ϵ with probability at least $1 - \delta$ in space $O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$.

3 Algorithms for learning from data streams

The ultimate goal of data mining is to develop systems and algorithms with high level of autonomy. For such, data mining studies the automated acquisition of domain knowledge with the goal of improving system's performance by learning from experience. These systems address the problems of data processing, modeling, prediction, clustering, and control in changing and evolving environments. They self-evolve their structure and knowledge on the environment. In this section, we review streaming algorithms for learning decision trees, clustering examples and frequent pattern mining.

3.1 Predictive learning from data streams

Hulten and Domingos [37] presents a general method to learn from arbitrarily large databases. The method consists of deriving an upper bound for the learner's loss as a function of the number of examples used in each step of the algorithm. Then use this to minimize the number of examples required at each step, while guaranteeing that the model produced does not differ significantly from the one that would be obtained with infinite data. This general methodology has been successfully applied in k-means clustering [37], Hierarchical clustering of variables [49], decision trees [25,38], etc.

Learning from large data sets may be more effective when using algorithms that place greater emphasis on bias management [29]. One such algorithms is the Very Fast Decision Tree system [25]. VFDT is a decision-tree learning algorithm that dynamically adjusts its bias whenever new examples are available. In decision-tree induction, the main issue is the decision of when to expand the tree, installing a splitting-test and generating new leaves. The basic idea consists of using a small set of examples to select the splitting-test to incorporate in a decision-tree node. If after seeing a set of examples, the difference of the merit between the two best splitting-tests does not satisfy a statistical test (the Hoeffding bound), VFDT proceeds by examining more examples. It only makes a decision (i.e., adds a splitting-test in that node), when there is enough statistical evidence in favor of a particular test. This strategy guarantees model stability (low variance), controls overfitting, while it may achieve an increased number of degrees of freedom (low bias) with increasing number of examples.

In VFDT a decision tree is learned by recursively replacing leaves with decision nodes. Each leaf stores the sufficient statistics about attribute-values. The sufficient statistics are those needed by a heuristic evaluation function that computes the merit of split-tests based on attribute-values. When an example is available, it traverses the tree from the root to a leaf, evaluating the appropriate attribute at each node, and following the branch corresponding to the attribute's value in the example. When the example reaches a leaf, the sufficient statistics are updated. Then, each possible condition based on attribute-values is evaluated. If there is enough statistical support in favor of one test over the others, the leaf is changed to a decision node. The new decision node will have as many descendant leaves as the number of possible values for the chosen attribute (therefore this tree is not necessarily binary). The decision nodes only maintain the information about the split-test installed within them.

The main innovation of the VFDT system is the use of Hoeffding bounds to decide how many examples must be observed before installing a split-test at a leaf. Suppose we

have made n independent observations of a random variable r whose range is R. The Hoeffding bound states that the true average of r, $\bar{r}$, is at least $\bar{r} - \epsilon$ where $\epsilon = \sqrt{R^2 \frac{\ln(\frac{1}{\delta})}{2n}}$, with probability $1 - \delta$.

Let H be the evaluation function of an attribute. For the information gain, the range R, of H is $\log_2(k)$ where k denotes the number of classes. Let x_a be the attribute with the highest H, x_b the attribute with second highest H and $\overline{\Delta H} = \overline{H}(x_a) - \overline{H}(x_b)$, the difference between the two best attributes. Then if $\overline{\Delta H} > \epsilon$ with n examples observed in the leaf, the Hoeffding bound states that, with probability $1 - \delta$, x_a is really the attribute with the highest value in the evaluation function. In this case, the leaf must be transformed into a decision node that splits on x_a.

VFTD has been extended to deal with continuous attributes [31], functional leaves [32] and non-stationary data streams in [38,30]. The CVFDT algorithm [38], an extension to VFDT designed for time-changing data streams. CVFDT generates alternative decision trees at nodes where there is evidence that the splitting test is no longer appropriate. The system replaces the old tree with the new one when the latter becomes more accurate. An interesting characteristic of VFDT is the ability to freeze less promising leaves, when working in memory-restricted environments. A VFDT like algorithm for learning regression and model trees appear in [39]. Bagging and boosting ensemble models, using VFDT like algorithms, appear in [14,13].

3.2 Clustering data streams

Clustering is the process of grouping objects into different groups, such that the common properties of data in each subset are high, and between different subsets are low. The data stream clustering problem is defined as *to maintain a continuously consistent good clustering of the sequence observed so far, using a small amount of memory and time*. The issues are imposed by the continuous arriving data points, and the need to analyze them in real time. These characteristics requires incremental clustering, maintaining cluster structures that evolve over time. Moreover, the data stream may evolve over time, and new clusters might appear, other disappears, reflecting the dynamics of the stream.

A powerful idea in clustering from data streams is the concept of *cluster feature (CF)*. A cluster feature, or *micro-cluster*, is a compact representation of a set of points. A CF structure is a triple (N, LS, SS), used to store the sufficient statistics of a set of points: where N is the number of data points; LS is a vector, of the same dimension of data points, that store the linear sum of the N points; SS is a vector, of the same dimension of data points, that store the square sum of the N points.

The properties of cluster features are:

- **Incrementality**
  If a point x is added to the cluster A, the sufficient statistics are updated as follows:

$$LS_A \leftarrow LS_A + x;\ SS_A \leftarrow SS_A + x^2;\ N_A \leftarrow N_A + 1$$

- **Additivity**
  if A and B are disjoint sets, merging them is equal to the sum of their parts. The additive property allows us to merge sub-clusters incrementally.

$$LS_C \leftarrow LS_A + LS_B;\ SS_C \leftarrow SS_A + SS_B;$$
$$N_C \leftarrow N_A + N_B.$$

A CF entry has sufficient information to calculate the norms L_1 and L_2 (See Eq. 1), and basic measures to characterize a cluster.

$$L_1 = \sum_{i=1}^{n} |x_{a_i} - x_{b_i}|;\ L_2 = \sqrt{\sum_{i=1}^{n} (x_{a_i} - x_{b_i})^2} \quad (1)$$

The idea of dividing the clustering process into two layers, where the first layer generate local models (micro-clusters) and the second layer generates global models from the local ones, is a powerful idea that has been used elsewhere.

The BIRCH system [55] builds a hierarchical structure of data, the CF-tree, where each node contains a set of cluster features. These CF's contain the sufficient statistics describing a set of points in the data set, and all information of the cluster features below in the tree. The system requires two user defined parameters: B the branch factor or the maximum number of entries in each non-leaf node; and T the maximum diameter (or radius) of any CF in a leaf node. The maximum diameter T defines the examples that can be *absorbed* by a CF. Increasing T, more examples can be absorbed by a micro-cluster and smaller CF-Trees are generated.

When an example is available, it traverses down the current tree from the root, till finding the appropriate leaf. At each non-leaf node, the example follow the *closest*-CF path, with respect to norms L_1 or L_2. If the closest-CF in the leaf cannot absorb the example, make a new CF entry. If there is no room for new leaf, split the parent node. A leaf node might be expanded due to constrains imposed by B, and T. The process consists of taking the two farthest CFs and creates two new leaf nodes. When traversing backup the CFs are updated.

The CluStream Algorithm [3] is an extension of the BIRCH system designed for data streams. Here, the CFs includes temporal information: the time-stamp of an example is treated as a feature. CFs are initialized offline, using a

standard k-means, with a large value for k. For each incoming data point, the distance to the centroids of existing CFs, are computed. The data point is absorbed by an existing CF if the distance to the centroid falls within the *maximum boundary* of the CF. The *maximum boundary* is defined as a factor t of the *radius* deviation of the CF; Otherwise, the data point starts a new micro-cluster.

`CluStream` can generate approximate clusters for any user defined time granularity. This is achieved by storing the CFT at regular time intervals, referred to as snapshots. Suppose the user wants to find clusters in the stream based on a history of length h. The offline component can analyze the snapshots stored at the snapshots t, the current time, and $(t - h)$ using the addictive property of CFT. An important problem is when to store the snapshots of the current set of micro-clusters. For example, the natural time frame stores snapshots each quarter, four quarters are aggregated in hours, 24 h are aggregated in days, etc. The aggregation level is domain-dependent and explores the addictive property of CFT.

3.3 Frequent pattern mining

Since their introduction in [4], the frequent itemset (and association rule) mining problems have received a great deal of attention. Within the past decade, hundreds of research papers have been published presenting new algorithms or improvements on existing algorithms to solve these mining problems more efficiently. The `Apriori` [4] level wise approach implies several scans over the database to compute the support of candidate frequent itemsets. As an alternative, several algorithms significantly reduce this by generating collections of candidate itemsets in a depth-first strategy. The reference algorithm in this line is the `FP-growth` algorithm by [36]. It uses a prefix-tree (a trie) to store itemsets. To generate all possible extensions of an itemset by a single item, it simply appends the item to the suffix-tree. It avoids the self-joins required in `Apriori` for candidate generation. This search scheme generates each candidate itemset at most once. It has been used as a building block in frequent itemsets and sequence mining from data streams.

Mining frequent itemsets from data streams poses many new challenges. In addition to the one-scan constraint, the limited memory requirement, the combinatorial explosion of itemsets exacerbates the difficulties. The most difficult problem in mining frequent itemsets from data streams is that infrequent itemsets in the past might become frequent, and frequent itemsets in the past might become infrequent.

The `FP-tree` algorithm was used as a building block for mining frequent patterns in data streams at multiple time granularities in [34]. The `FP-Stream Algorithm` was designed to maintain frequent patterns under a tilted-time window framework to answer time-sensitive queries. The frequent patterns are compressed and stored using a tree structure similar to FP-tree and updated incrementally with incoming transactions. Quoting [34]:

> *Using this scenario, we can answer the following queries: (1) what is the frequent pattern set over the period t_2 and t_3? (2) what are the periods when $\{a, b\}$ is frequent? (3) does the support of $\{a\}$ change dramatically in the period from t_3 to t_0? and so on. That is, one cans (1) mine frequent patterns in the current window, (2) mine frequent patterns over time ranges with granularity confined by the specification of window size and boundary, (3) put different weights on different windows to mine various kinds of weighted frequent patterns, and (4) mine evolution of frequent patterns based on the changes of their occurrences in a sequence of windows.*

Time windows are a standard approach to deal with evolving data. The frequency of a pattern in different time windows also evolves, that is a pattern that was not frequent in the past might become frequent and vice-versa. To ensure the completeness of frequent patterns, [34] consider three categories of patterns: *frequent patterns*, *subfrequent patterns*, and *infrequent patterns*. The frequency of an itemset I over a period of time T is the number of transactions in T in which I occurs. The support of I is the frequency divided by the total number of transactions observed in T. Let the $min\_support$ be σ and consider a relaxation ratio $\rho = \epsilon/\sigma$, where ϵ is the maximum support error. I is frequent if its support is no less than σ; it is sub-frequent if its support is less than σ but not less than ρ; otherwise, it is infrequent.

The `FP-stream` structure consists of two parts. A global FP-tree held in main memory, and tilted-time windows embedded in this pattern-tree. Incremental updates can be performed on both parts of the `FP-stream`. Incremental updates occur when some infrequent patterns become (sub) frequent, or vice versa. At any moment, the set of frequent patterns over a period can be obtained from FP-stream.

`FP-stream` stores the frequencies for itemset I in a tilted-time window[1]. Assume that the stream of transactions is broken up into batches $B_1, B_2, \ldots, B_n, \ldots$ of fixed sized, where B_n is the most current batch and B_1 the oldest.

As the transactions of the first batch B_1 arrived, the frequencies for all the items are computed, and an ordering f_list is created, as in the `FP-tree` Algorithm. This order remains fixed for the subsequent batches. The transactions of B_1 are processed again creating an `FP-tree` pruning all items with frequency less than $\epsilon \times |B_1|$.

The maintenance of the tilted-time windows is straightforward. When four quarters are accumulated, they are merged

[1] Giannella et al. [34] discusses also a more compact structure using logarithmic tilted-time windows.

together in 1 h bin. After 24 h, 1 day is built, and so on. This model allows to compute the frequent itemsets in the last hour with the precision of a quarter of an hour, the last day frequent itemsets with a precision of an hour, the last month with a precision of a day, etc. For a period of 1 month we need $4 + 24 + 31 = 59$ units of time. Let $t_1, \ldots, t_n$ be the tilted-time windows which group the batches seen so far. Denote the number of transactions in t_i by w_i. The goal is to mine all frequent itemsets with support larger than σ over period $T = t_k \cup t_{k+1} \cup \ldots \cup t_{k'}$, where $1 \leq k \leq k' \leq n$. The size of T, denoted by W, is the sum of the sizes of all time-windows considered in T. It is not possible to store all possible itemsets in all periods. `FP-stream` drops the tail sequences when $\forall_i, n \leq i \leq 1$, $f_I(t_i) < \sigma \times w_i$ and $\sum_{j=n}^{i} f_I(t_j) < \epsilon \times \sum_{j=n}^{i} w_j$. We no longer have the exact frequencies over T. By delivering all frequent itemsets larger than $(\sigma - \epsilon) \times W$ any frequent itemset in T will not miss, although we might get itemsets whose frequency is between $(\sigma - \epsilon) \times W$ and $\sigma \times W$.

Itemsets and their tilted-time window tables are maintained in the `FP-stream` data structure. When a new batch B arrives, mine the itemsets from B and update the `FP-stream` structure. For each itemset I mined in B, if I does not appear in the structure, add I if $f_I(B) \geq \epsilon|B|$. Otherwise, add $f_I(B)$ to I's table and then do tail pruning. If all the windows were dropped, then drop I from the `FP-stream` data structure. Moreover, any superset of I will also be dropped.

4 Algorithm issues in learning from data streams

The challenge problem for data mining is the ability to permanently maintain an accurate decision model. This issue requires learning algorithms that can modify the current model whenever new data are available at the rate of data arrival. Moreover, they should forget older information when data are outdated. In this context, the assumption that examples are generated at random according to a stationary probability distribution does not hold, at least in complex systems and for large periods of time. In the presence of a non-stationary distribution, the learning system must incorporate some form of forgetting past and outdated information. Learning from data streams require incremental learning algorithms that take into account concept drift. Solutions to these problems require new sampling and randomization techniques, and new approximate, incremental and decremental algorithms. Hulten and Domingos [37] identifies desirable properties of learning systems that are able to mine continuous, high-volume, open-ended data streams as they arrive. Learning systems should be able to process examples and answering queries at the rate they arrive. Some desirable properties for learning in data streams include: incrementality, online learning, constant time to process each example, single scan over the training set and taking drift into account.

4.1 Cost-performance management

Incremental learning is one fundamental aspect for the process of continuous adaptation of the decision model. The ability to update the decision model, whenever new information is available, is an important property, but it is not enough as it also requires operators with the ability to *forget* past information [43]. Some data stream models allow delete and update operators. Sliding window models require forgetting old information. In all these situations, the incremental property is not enough. Learning algorithms need forgetting operators that reverse learning: decremental unlearning [16].

The incremental and decremental issues require a permanent maintenance and updating of the decision model as new data are available. Of course, there is a trade-off between the cost of update and the gain in performance that we may obtain. Learning algorithms exhibit different profiles. Algorithms with strong variance management are quite efficient for small training sets. Very simple models, using a few free-parameters, can be quite efficient in variance management, and effective in incremental and decremental operations being a natural choice in the sliding windows framework. The main problem with simple representation languages is the boundary in generalization performance they can achieve, since they are limited by high bias while large volumes of data require efficient bias management. Complex tasks requiring more complex models increase the search space and the cost for structural updating. These models, require efficient control strategies for the trade-off between the gain in performance and the cost of updating. A step in this direction is the so-called *algorithm output granularity* presented by [27]. Algorithm output granularity monitors the amount of mining results that fits in main memory before any incremental integration. Gaber et al. [28] illustrates the application of the *algorithm output granularity* strategy to build efficient clustering, frequent items and classification techniques.

In most applications, we are interested in maintaining a decision model consistent with the current status of the nature. This lead us to the sliding window models where data are continuously inserted and deleted from a window. Learning algorithms must have operators for incremental learning and forgetting. Incremental learning and forgetting are well defined in the context of predictive learning. The meaning or the semantics in other learning paradigms (like clustering) are not so well understood, very few works address this issue.

4.2 Monitoring learning

When data flow over time, and at least for large periods of time, it is highly unprovable the assumption that the examples

are generated at random according to a stationary probability distribution. At least in complex systems and for large time periods, we should expect changes in the distribution of the examples. A natural approach for these *incremental tasks* are *adaptive learning algorithms*, incremental learning algorithms that take into account concept drift.

Concept drift means that the concept related to the data being collected may shift from time to time, each time after some minimum permanence. Changes occur over time. The evidence for changes in a concept are reflected in some way in the training examples. Old observations, that reflect the past behavior of the nature, become irrelevant to the current state of the phenomena under observation and the learning agent must forget that information.

The nature of change is diverse. It might occur, in the context of learning, due to changes in hidden variables, or changes in the characteristic properties of the observed variables.

Most learning algorithms use blind methods that adapt the decision model at regular intervals without considering whether changes have really occurred. Much more interesting is explicit change detection mechanisms. The advantage is that they can provide meaningful description (indicating change-points or small time-windows where the change occurs) and quantification of the changes. They may follow two different approaches:

1. Monitoring the evolution of performance indicators adapting techniques used in statistical process control.
2. Monitoring distributions on two different time windows.

The main research issue is how to incorporate change detection mechanisms in the learning algorithm, embedding change detection methods in the learning algorithm is a requirement in the context of continuous flow of data. The level of *granularity* of decision models is a relevant property, because if can allow partial, fast and efficient updates in the decision model instead of rebuilding a complete new model whenever a change is detected. The ability to recognize seasonal and re-occurring patterns is an open issue.

Concept drift in the predictive classification setting is a well-studied topic. In other learning scenarios, like clustering, very few works address the problem. The main research issue is how to incorporate change detection mechanisms in the learning algorithm for different paradigms.

4.3 Novelty detection

Novelty detection refers to learning algorithms being able to identify and learn new concepts. Intelligent agents that act in dynamic environments must be able to learn conceptual representations of such environments. Those conceptual descriptions of the world are always incomplete, they correspond to what it is *known* about the world. This is the *open* world assumption as opposed to the traditional *closed* world assumption, where what is to be learnt is defined in advance. In open worlds, learning systems should be able to extend their representation by learning new concepts from the observations that do not match the current representation of the world. This is a difficult task. It requires to identify the *unknown*, i.e., the limits of the current model. In that sense, the *unknown* corresponds to an *emerging pattern* that is different from *noise*, or *drift* in previously known concepts.

4.4 Distributed streams

Data streams are distributed in nature. Learning from distributed data, we need efficient methods in minimizing the communication overheads between nodes [50].

The strong limitations of centralized solutions is discussed in depth in [40,41]. The authors point out *a mismatch between the architecture of most off-the-shelf data mining algorithms and the needs of mining systems for distributed applications*. Such mismatch may cause a bottleneck in many emerging applications, namely hardware limitations related to the limited bandwidth channels. Most importantly, in applications like monitoring, centralized solutions introduce delays in event detection and reaction, that can make mining systems useless.

Another direction, for distributed processing, explore multiple models [19,42]. Kargupta et al. [42] proposes a method that offers an effective way to construct a redundancy-free, accurate, and meaningful representation of large decision-tree ensembles often created by popular techniques such as bagging, boosting, random Forests and many distributed and data stream mining algorithms.

4.5 Structured data

In some challenging applications of data mining, data are better described by sequences (for example DNA data), trees (XML documents), and graphs (chemical components). Tree mining in particular is an important field of research [11,12]. XML patterns are tree patterns, and XML is becoming a standard for information representation and exchange over the Internet; the amount of XML data is growing, and it will soon constitute one of the largest collections of human knowledge.

4.6 Evolving feature spaces

In the static case, similar data can be described with different schemata. In the case of dynamic streams, the schema of the stream can also change. We need algorithms that can deal with evolving feature spaces over streams. There is very little work in this area, mainly pertaining to document streams. For

example, in sensor networks, the number of sensors is variable (usually increasing) over time.

4.7 Evaluation methods and metrics

An important aspect of any learning algorithm is the hypothesis evaluation criteria. Most of evaluation methods and metrics were designed for the static case and provide a single measurement about the quality of the hypothesis. In the streaming context, we are much more interested in how the evaluation metric evolves over time. Results from the *sequential statistics* [54] may be much more appropriate. Gama et al. [33] proposes a general framework for assessing predictive stream learning algorithms using sequential statistics. They show that the prequential error converges to an holdout estimator when computed over sliding windows or using fading factors.

5 Emerging challenges and future issues

In a recent paper, Muthukrishnan [47] identifies the main challenges in data stream management systems: computational models for massive distributed data, continual computation theory and stochastic data algorithms. Current developments in these directions include algorithms and computational models for monitoring TCP/IP networks [22]; to compute evolving profiles from telecommunications traffic [23]; storing, querying and mining scientific data in the virtual telescope project [52]; indexing and mining web data for improving search engines [9], etc. From a data mining perspective, there is a fundamental difference between learning from small data sets and large data sets. As pointed-out by some researchers [15], current learning algorithms emphasize variance reduction. However, learning from large data sets may be more effective when using algorithms that place greater emphasis on bias management.

In another dimension, simple objects that surround us are changing from static, inanimate objects into adaptive, reactive systems with the potential to become more and more useful and efficient. Smart things associated with all sort of networks offers new unknown possibilities for the development and self-organization of communities of intelligent communicating appliances. The dynamic characteristics of data flowing over time requires adaptive algorithms. While the languages used to represent generalizations from examples are well understood, next generation data mining algorithms should care, at least, about the cost-performance management, and the limitations in all aspects of computational resources. Learning algorithms must be able to *adapt continuously to changing environmental conditions* (including their own condition) and evolving user needs. Learning must consider the *real-time constrains of limited computer, battery power and communication resources*.

Intelligent agents that adapt over time in a dynamic and sometimes in adversary conditions, should be capable of *self-diagnosis*. A significant and useful intelligence characteristic is diagnostics—not only after failure has occurred, but also predictive (before failure) and advisory (providing maintenance instructions). The development of such self-configuring, self-optimizing, and self-repairing systems is a major scientific and engineering challenge. All these aspects requires monitoring the evolution of learning process itself, and the ability of reasoning and learning about it.

Acknowledgements This work was supported by research project *Knowledge Discovery from Ubiquitous Data Streams* (PTDC/EIA-EIA/098355/2008).

References

1. Aggarwal, C.: On biased reservoir sampling in the presence of stream evolution. In: Dayal, U., Whang, K.-Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.-K. (eds.) Proceedings of the International Conference on Very Large Data Bases, pp. 607–618. ACM Seoul, Korea (2006)
2. Aggarwal, C. (ed): Data Streams—Models and algorithms. Springer, Berlin (2007)
3. Aggarwal, C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: Proceedings of the International Conference on Very Large Data Bases, pp. 81–92. Morgan Kaufmann, Berlin (2003)
4. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 207–216. Washington, DC, USA (1993)
5. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. J. Comput. Syst. Sci. **58**, 137–147 (1999)
6. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Kolaitis, P.G. (ed.) Proceedings of the 21st Symposium on Principles of Database Systems, pp. 1–16. ACM Press, Madison (2002)
7. Babcock, B., Datar, M., Motwani, R.: Sampling from a moving window over streaming data. In: Proceedings of the Annual ACM SIAM Symposium on Discrete Algorithms, pp. 633–634. Society for Industrial and Applied Mathematics, San Francisco (2002)
8. Babu, S., Widom, J.: Continuous queries over data streams. SIGMOD Rec. **30**(3), 109–120 (2001)
9. Baeza-Yates, R.A., Broder, A.Z., Maarek, Y.S.: The new frontier of web search technology, Seven challenges. In: SeCO Workshop. Lecture Notes in Computer Science, vol. 6585, pp. 3–9. Springer, Berlin (2010)
10. Bifet, A., Gavaldà, R.: Kalman filters and adaptive windows for learning in data streams. In: Todorovski, L., Lavrac, N. (eds.) Proceedings of the 9th Discovery Science, Lecture Notes Artificial Intelligence, vol. 4265, pp. 29–40. Springer, Barcelona (2006)
11. Bifet, A., Gavaldà, R.: Mining adaptively frequent closed unlabeled rooted trees in data streams. In: Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, pp. 34–42. Las Vegas, USA (2008)
12. Bifet, A., Gavaldà, R.: Adaptive XML tree classification on evolving data streams. In: Machine Learning and Knowledge Discovery

in Databases, European Conference, Lecture Notes in Computer Science, vol. 5781, pp. 147–162. Springer, Bled (2009)
13. Bifet, A., Holmes, G., Pfahringer, B.: Leveraging bagging for evolving data streams. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML/PKDD (1), Lecture Notes in Computer Science, vol. 6321, pp. 135–150. Springer, Berlin (2010)
14. Bifet, A., Holmes, G., Pfahringer, B., Gavaldà, R.: Improving adaptive bagging methods for evolving data streams. In: Zhou, Z.-H., Washio, T. (eds.) ACML, Lecture Notes in Computer Science, vol. 5828, pp. 23–37. Springer, Berlin (2009)
15. Brain, D., Webb, G.: The need for low bias algorithms in classification learning from large data sets. In: Elomaa, T., Mannila, H., Toivonen, H (eds.) Principles of Data Mining and Knowledge Discovery PKDD-02, Lecture Notes in Artificial Intelligence, vol. 2431, pp. 62–73. Springer, Helsinki (2002)
16. Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. In: Proceedings of the Neural Information Processing Systems (2000)
17. Chakrabarti, A., Ba, K.D., Muthukrishnan, S.: Estimating entropy and entropy norm on data streams. In: STACS: 23rd Annual Symposium on Theoretical Aspects of Computer Science, pp.196–205. Marseille, France (2006)
18. Chaudhry, N.: Stream Data Management, Chapter Introduction to Stream Data Management, pp. 1–11. Springer, Berlin (2005)
19. Chen, R., Sivakumar, K., Kargupta, H.: Collective mining of Bayesian networks from heterogeneous data. Knowl. Inform. Syst. J. **6**(2), 164–187 (2004)
20. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithm **55**(1), 58–75 (2005)
21. Cormode, G., Muthukrishnan, S., Zhuang, W.: Conquering the divide: Continuous clustering of distributed data streams. In: ICDE: Proceedings of the International Conference on Data Engineering, pp. 1036–1045. Istanbul, Turkey (2007)
22. Cormode, G., Thottan, M. (eds.): Algorithms for Next Generation Networks. Springer, Berlin (2010)
23. Cortes, C., Fisher, K., Pregibon, D., Rogers, A., Smith, F.: Hancock: a language for analyzing transactional data streams. ACM Trans. Progr. Languages Syst. **26**(2), 301–338 (2004)
24. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. In: Proceedings of Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, pp. 635–644. Springer, San Francisco (2002)
25. Domingos, P., Hulten, G.: Mining High-Speed Data Streams. In: Parsa, I., Ramakrishnan, R., Stolfo, S. (eds.) Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining, pp. 71–80. ACM Press, Boston (2000)
26. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J Comput. Syst. Sci. **31**(2), 182–209 (1985)
27. Gaber, M. M., Yu, P.S.: A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering. In: ACM Symposium Applied Computing, pp. 649–656. ACM Press, Boston (2006)
28. Gaber, M.M., Krishnaswamy, S., Zaslavsky, A.: Cost-efficient mining techniques for data streams. In: Proceedings of the second workshop on Australasian information security, pp. 109–114. Australian Computer Society, Inc., Melbourne (2004)
29. Gama, J.: Knowledge Discovery from Data Streams. Data Mining and Knowledge Discovery. Chapman & Hall/CRC Press, Atlanta (2010)
30. Gama, J., Fernandes, R., Rocha, R.: Decision trees for mining data streams. Intell. Data Anal. **10**(1), 23–46 (2006)
31. Gama, J., Medas, P.: Learning decision trees from dynamic data streams. J. Univers. Comput. Sci. **11**(8), 1353–1366 (2005)
32. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 523–528. ACM Press, Washington, DC (2003)
33. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: KDD, pp. 329–338 (2009)
34. Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.: Mining frequent patterns in data streams at multiple time granularities. In: Kargupta, H., Joshi, A., Sivakumar, K., Yesha, Y. (eds.) Data Mining: Next Generation Challenges and Future Directions, pp. 105–124. AAAI/MIT Press, Cambridge (2004)
35. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In: VLDB, pp. 79–88. Rome, Italy (2001)
36. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation. Data Min. Knowl. Discov. **8**, 53–87 (2004)
37. Hulten, G., Domingos, P.: Catching up with the data: research issues in mining data streams. In: Proceedings of Workshop on Research Issues in Data Mining and Knowledge Discovery, Santa Baraba, USA (2001)
38. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 97–106. ACM Press, San Francisco (2001)
39. Ikonomovska, E., Gama, J., Džeroski, S.: Learning model trees from evolving data streams. Data Min. Knowl. Discov. **23**, 128–168 (2011). doi:10.1007/s10618-010-0201-y
40. Kargupta, H., Joshi, A., Sivakumar, K., Yesha, Y.: Data Mining: Next Generation Challenges and Future Directions. AAAI Press and MIT Press, Cambridge (2004)
41. Kargupta, H., Park, B.-H.: Mining decision trees from data streams in a mobile environment. In: IEEE International Conference on Data Mining, pp. 281–288. IEEE Computer Society, San Jose (2001)
42. Kargupta, H., Park, B.-H., Dutta, H.: Orthogonal decision trees. IEEE Trans. Knowl. Data Eng. **18**, 1028–1042 (2006)
43. Kifer, D., Ben-David, S., Gehrke, J.: Detecting change in data streams. In: Proceedings of the International Conference on Very Large Data Bases, pp. 180–191. Morgan Kaufmann, Toronto (2004)
44. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of 28th International Conference on Very Large Data Bases, pp. 346–357. Morgan Kaufmann, Hong Kong (2002)
45. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1997)
46. Muthukrishnan, S.: Data Streams: Algorithms and Applications. Now Publishers, USA (2005)
47. Muthukrishnan, S.: Massive data streams research: Where to go. Tech. Rep., Rutgers University (2010)
48. Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, Inc., San Mateo (1993)
49. Rodrigues, P.P., Gama, J., Pedroso, J.P.: Hierarchical clustering of time series data streams. IEEE Trans. Knowl. Data Eng. **20**(5), 615–627 (2008)
50. Sharfman, I., Schuster, A., Keren, D.: A geometric approach to monitoring threshold functions over distributed data streams. ACM Trans. Database Syst. **32**(4), 301–312 (2007)
51. Tatbul, N., Cetintemel, U., Zdonik, S., Cherniack, M., Stonebraker, M.: Load shedding in a data stream manager. In: Proceedings of the International Conference on Very Large Data Bases, pp. 309–320. VLDB Endowment, Berlin (2003)
52. Thakar, A.R., Szalay, A.S., Fekete, G., Gray, J.: The catalog archive server database management system. Comput. Sci. Eng. **10**(1), 30–37 (2008)

53. Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**(1), 37–57 (1985)
54. Wald, A.: Sequential Analysis. John Wiley and Sons, Inc., New York (1947)
55. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 103–114. ACM Press, Montreal (1996)