Data Streams Classification: A Selective Ensemble with Adaptive Behavior

Valerio Grossi[1] and Franco Turini[2]

[1] Dept. of Pure and Applied Mathematics, University of Padova
via Trieste 63, 35121 Padova, Italy
[2] Dept. of Computer Science, University of Pisa
largo B. Pontecorvo 3, 56127 Pisa, Italy
grossi@math.unipd.it, turini@di.unipi.it

Abstract. Data streams classification represents an important and challenging task for a wide range of applications. The diffusion of new technologies, such as smartphones and sensor networks, related to communication services introduces new challenges in the analysis of streaming data. The latter requires the use of approaches that require little time and space to process a single item, providing an accurate representation of only relevant data characteristics for keeping track of concept drift. Based on these premises, this paper introduces a set of requirements related to the data streams classification proposing a new adaptive ensemble method. The outlined system employs two distinct structure, for managing both data aggregation and mining features. The latter are represented by a selective ensemble managed with an adaptive behavior. Our approach dynamically updates the threshold value for enabling the models directly involved in the classification step. The system is conceived to satisfy the proposed requirements even in the presence of concept drifting events. Finally, our method is compared with several existing systems employing both synthetic and real data.

1 Introduction

The constant and rapid diffusion of new technologies, such as smartphones, and sensor networks, related to communication and web services, and safety applications, has introduced new challenges in data management, analysis and mining. In these scenarios data arrives on-line, at a time-varying rate creating the so-called data stream phenomenon. Conventional knowledge discovery tools cannot manage this overwhelming volume of data. The unpredictable nature of data streams requires the use of new approaches, which involve at most one pass over the data, and try to keep track of time-evolving features, known as concept drift.

Ensemble approaches represent a valid solution for data streams classification [5]. In these methods, classification takes advantage of multiple classifiers, extracting new models from scratch and deleting the out-of-date ones continuously. In [20,19], it was stressed that the number of classifiers actually involved in the classification task cannot be constant through time. The cited works demonstrate that a selective ensemble which, based on current data distribution, dynamically calibrates the set of classifiers to use, provides a better performance than systems using a fixed set of classifiers constant through time. In our former approach, the selection of the models involved in the classification step was chosen by a fixed activation threshold. This choice is the right solution if it is possible to study a-priori what is the best value to assign to the threshold. In many real environment, this information is unavailable, since the stream data behavior cannot be modeled. In several domains, such as intrusion detection, data distribution can remain stable for a long time, changing radically when an attack occurs.

This work presents an evolution of the system outlined in [20,19]. The new approach introduces a complete adaptive behavior in the management of the threshold required for the selection of the set of models actually involved in the classification. This work describes the adaptive approach for varying the value of the model activation threshold through time, influencing the overall behavior of the ensemble classifier, based on data change reaction. Our approach is explicitly explained with the use of binary attributes. This choice can be seen as a limitation, but it is worth observing that every nominal attribute can be easily transformed into a set of binary ones. The only inability is the direct treatment of numerical values. [14] represents a general approach to solve the on-line discretization of numerical attributes. The proposed method is particularly suitable in our context, since it proposes a discretization method based on two layers. The first layer summarizes data, while the second one constructs the final binning. The process of updating the first layer works on-line and requires a single scan over the data.

Paper Organization: Section 2 introduces our reference scenario, outlining some requirements that a system working on streaming environments should satisfy. Section 3 describes our approach in details, highlighting how the requirements introduced in Section 2.1 are verified by the proposed model. Furthermore, it present how our adaptive selection is implemented. Section 4 presents a comparative study to understand how the new adaptive approach guarantees a higher reliability of the system. In this section, our approach is compared with other well-know approaches available in the literature. Finally, Section 5 draws the conclusions and introduces some future works.

2 Data Streams Classification

Data streams represent a new challenge for the data mining community. In a stream scenario, traditional mining methods are further constrained by the unpredictable behavior of a large volume of data. The latter arrives on-line at variable rates, and once an element has been processed, it must be discarded or archived. In either cases, it cannot be easily retrieved. Mining systems have no control over data generation, and they must be capable of guaranteeing a near real-time response.

Definition 1. *A data stream is an infinite set of elements* $X = X_1,\ldots,X_j,\ldots$ *where each* $X_i \in X$ *has* $a+1$ *dimensions,* $(x_i^1,\ldots x_i^a,y)$*, and where* $y \in \{\perp,1,\ldots,C\}$*, and* $1,\ldots,C$ *identify the possible values in a class.*

A stream can be divided into two sets based on the availability of class label y. If value y is available in the record ($y \neq \perp$), it belongs to the training set. Otherwise the record represents an element to classify, and the true label will only be available after an unpredictable period of time.

Given Definition 1, the notion of *concept drift* can be easily defined. As reported in [23], a data stream can be divided into batches, namely $b_1, b_2, ..., b_n$. For each batch b_i, data is independently distributed w.r.t. a distribution $P_i()$. Depending on the amount and type of concept drift, $P_i()$ will differ from $P_{i+1}()$. A typical example is customers' buying preferences, which may change according to the day of the week, inflation rate and/or availability of alternatives. Two main types of concept drift are usually distinguished in the literature, i.e. *abrupt* and *gradual*. Abrupt changes imply a radical variation of data distribution from a given point in time, while gradual changes are characterized by a constant variation during a period of time. The concept drifting phenomenon involves data expiration directly, forcing stream mining systems to be continuously updated to keep track of changes. This implies making time-critical decisions for huge volumes of high-speed streaming data.

2.1 Requirements

As introduced in Section 2, the stream features influence the development of a data streams classifier radically. A set of requirements must be taken into account before proposing a new approach. These needs highlight several implementation decisions inserted in our approach.

Since data streams can be potentially unbounded in size, and data arrives at unpredictable rates, there are rigid constraints on time and memory required by a system through time:

Req. 1: the time required for processing every single stream element must be constant, which implies that every data sample can be analyzed almost only once.

Req. 2: the memory needed to store all the statistics required by the system must be constant in time, and it cannot be related to the number of elements analyzed.

Req. 3: the system must be capable of updating their structures readily, working within a limited time span, and guaranteeing an acceptable level of reliability.

Given Definition 1, the elements to classify can arrive in every moment during the data flow.

Req. 4: the system must be able to classify unseen elements every time during its computation.

Req. 5: the system should be able to manage a set of models that do not necessarily include contiguous ones, i.e. classifiers extracted using adjacent parts of the stream.

2.2 Related Work

Mining data streams has rapidly become an important and challenging research field. As proposed in [12], the available solutions can be classified into *data-based* and *task-based* ones. In the former approaches a data stream is transformed into an approximate smaller-size representation, while task-based techniques employ methods from computational theory to achieve time and space efficient solutions. *Aggregation* [1,2,3], *sampling* [10] or *summarized data structure*, such as histograms [21,17], are popular example of data-based solutions. On the contrary, *approximation algorithms* such as those introduced in [15,10] are examples of task-based techniques.

In the context of data streams classification, two main approaches can be outlined, namely *instance selection* and *ensemble learning*. Very Fast Decision Trees (VFDT) [9] with its improvements [22,14,27] for concept drifting reaction and numerical attributes managing represent examples of instance selection methods. In particular, the Hoeffding bound guarantees that the split attribute chosen using n examples, is the same with high probability as the one that would be chosen using an infinite set of examples. Last et al. [8] propose another strategy using an info-fuzzy technique to adjust the size of a data window. Ensemble learning employs multiple classifiers, extracting new models from scratch and deleting the out-of-date ones continuously. On-line approaches for bagging and boosting are available in [26,7,5]. Different methods are available in [30,34,29,25,11], where an ensemble of weighted-classifiers, including an adaptive genetic programming boosting, as in [11], is employed to cope with concept drifting. None of the two techniques can be assumed to be more appropriate than the other. [5] provides a comparison between different techniques not only in terms of accuracy, but also including computational features, such as memory and time required by each system. By contrast, our approach proposes an ensemble learning that differs from the cited methods since it is designed to concurrently manage different sliding windows, enabling the use of a set of classifiers not necessarily contiguous and constant in time.

3 Adaptive Selective Ensemble

A detailed description of our system is available in [19,18]. In the following subsections, we introduce only the main concepts of our approach highlighting the relations between the requirements outlined in Section 2.1 and the aggregate structures introduced. The proposed structures are primarily conceived to capture evolving data features, and guarantee data reduction at the same time. Ensuring a good trade-off between data reduction, and a powerful representation of all the evolving data factors is a non-trivial task.

3.1 The Snapshot

The snapshot definition implies the naïve Bayes classifier directly. In our model, the streaming training set is partitioned into chunks. Each data chunk is transformed into an approximate more compact form, called *snapshot*.

Definition 2 (Snapshot). *Given a data chunk of k elements, with A attributes and C class values, a snapshot computes the distribution of the values of attribute $a \in A$ with class value c, considering the last k elements arrived:*

$$S_k : C \times A \mapsto freq\big(a,k,c\big),\ \forall a \in A, c \in C$$

The following properties are directly derived from Definition 2.

Property 1. Given a stream with C class values and A attributes, a snapshot is a set of $C \times A$ tuples.

x y z	*class value*
0 1 1	no
0 1 0	yes
0 1 1	no
1 1 1	yes
1 0 0	yes
1 0 1	no
1 1 1	yes
0 0 0	ind
0 0 1	yes
0 0 0	ind

(a) A stream chunk of 10 elements.

$$\Rightarrow \; S_{10} = \left\| \begin{array}{c|c|c} (x,2,3) & (y,2,3) & (z,3,2) \\ (x,2,1) & (y,2,1) & (z,0,3) \\ (x,2,0) & (y,2,0) & (z,2,0) \end{array} \right\| \begin{array}{c} \texttt{yes} \\ \texttt{no} \\ \texttt{ind} \end{array}$$

(b) The resulting snapshot.

Fig. 1. From data stream (a) to snapshot (b)

$$\underbrace{\underbrace{\underbrace{e_1^j,\dots,e_k^j}_{S_{k,j}}\;\underbrace{e_1^{j+1},\dots,e_k^{j+1}}_{S_{k,j+1}}}_{S^2_{2k,j}}\;\underbrace{\underbrace{e_1^{j+2},\dots,e_k^{j+2}}_{S_{k,j+2}}\;\underbrace{e_1^{j+3},\dots,e_k^{j+3}}_{S_{k,j+3}}}_{S^2_{2k,j+1}}}_{S^3_{4k,j}} \cdots \underbrace{e_1^n,\dots,e_k^n}_{S_{k,n}}\;\underbrace{e_1^{n+1},\dots,e_k^{n+1}}_{S_{k,n+1}} \cdots$$

Fig. 2. Snapshots and their order

Property 2. Building a snapshot S_k requires k accesses to the data stream. Every element is accessed only once. Computing a snapshot is linear to the k number of elements.

Figure 1 shows an example of snapshot creation. The latter implies only a single access to every stream element. A snapshot is built incrementally accessing the data one by one, and updating the respective counters. Properties 1 and 2 guarantee that a snapshot requires a constant time and memory space, satisfying Requirements 1 and 2.

Snapshots of Higher Order. The only concept of snapshot is not sufficient to guarantee all the features needed for data managing and drift reaction. The concept of high-order snapshot is necessary to maximize data availability for the mining task guaranteeing only one data access.

Definition 3 (High-order Snapshot). *Given an order value $i > 0$, a high-order snapshot, is obtained by summing h snapshots of $i-1$ order:*

$$S^i_{h\times k} = \sum_{j=1}^{h} S^{i-1}_{k,j}$$

$$\sum_{j=1}^{h} \left[freq_j\big(a,k,c\big)\right]^{i-1},\ \forall a \in A, c \in C$$

where, given a class value c and an attribute a, $\left[freq_j\left(a,k,c\right)\right]^{i-1}$ *refers to the distribution of the values of attribute a with class value c of the j-th snapshot of order* $i-1$.

Figure 2 shows the relation between snapshots and their order. The aim is to employ a set of snapshots created directly from the stream to build new ones, representing increasingly larger data windows, simply by summing the frequencies of their elements.

A high-order snapshot satisfies Property 1, since it has the same structure of a basic one. Moreover, it further verifies Requirements 3, since the creation of a new high-order snapshot is linear in the number of attributes and class values. The creation of high-order snapshots does not imply any loss of information. This aspect guarantees that a set of different size sliding windows is simultaneously managed by accessing data stream only once, enabling the approach to consider every window as computed directly from the stream.

From a snapshot, or a high-order one, the system extracts an approximated decision tree, or employs the snapshot as naïve Bayes classifier directly.

3.2 The Frame

Snapshots are stored to maximize the number of elements for training classifiers. A model mined from a small set of elements tends to be less accurate than the one extracted from a large data set. If this observation is obvious in "traditional" mining contexts, where training sets are accurately built to maximize the model reliability, in a stream environment this is not necessarily true. Due to concept drifting, a model extracted from a large set of data can be less accurate than the one mined from a small training set. The large data set can include mainly out-of-date concepts.

Snapshots are then stored and managed, based on their order, in a structure called *Frame*. The order of a snapshot defines its level of time granularity. Conceptually similar to *Pyramidal Time Frame* introduced by Aggarwal et al. in [1] and inherited by *logarithmic tilted-time window*, our structure sorts snapshots based on the number of elements from which a snapshot was created.

Definition 4 (Frame). *Given a level value i, and a level capacity j, a frame is a function that, given a pair of indexes* (x,y) *returns a snapshot of order x and position y:*

$$F_{i,j} : (x,y) \mapsto Snapshot_{x,y}$$

where: $x \in \{0,\ldots,i-1\}$ *and* $y \in \{0,\ldots,j-1\}$.

As shown in Figure 3, level 1 contains snapshots created directly from the stream. Upper levels use the snapshots of the layer immediately lower to create a new one. The maximum number of snapshots available in the frame is constant in time and is defined by the number of levels and the level capacity. For each layer, the snapshot are stored with FIFO policy. The frame memory occupation is constant in time and is linear with the number of snapshots storable in the structure.

3.3 Ensemble Management

The concepts introduced in Section 3.1 and 3.2 are employed to define and manage an ensemble of classifiers. The selective ensemble management is defined by as a four phase approach:

1. For each snapshot S_i^j, a triple (C_i,w_i,b_i) representing the classifier, its weight and the classifier enabling variable b_i is extracted from S_i^j.
2. Since data distribution can change through time, the models currently in the structure are re-weighed with the new data distribution, using a test set of complete data taken from the last portion of the stream directly.
3. Given a level i, every time a new classifier is generated, the system decides if the new model must be inserted in the ensemble based on new data distribution. Apart from the lowest level, where the new one is inserted in any case, the system selects the k_i most promising models, based on the current weight associated to the classifiers, to classify the new data distribution from $k_i + 1$ models correctly.
4. Finally, a set of *active models* is selected, setting the boolean value b_i associated with a C_i as *true*. The set of *active models* is selected based on the value of an activation threshold θ. All the classifiers that differ at most θ to the best classifier with the highest weight are enabled.

The defined approach satisfies Requirements 4 and 5, since it can classify a new instance every time it is required, and can employ a set of not necessarily contiguous classifiers, since it is not necessarily true that every classifier generated through time enters the ensemble, but even in that case it can be disabled.

Figure 4 shows the overall organization of our system. For each level in the frame structure we have a corresponding level in the ensemble. The subdivision of the data aggregation task from the mining aspects makes our approach suitable in distributed/ parallel environments as well. One or more components can be employed to manage the concepts of snapshot and frame, while another can manage the ensemble classifier.

level			
1	$S_{k,j+2}$	$S_{k,j+1}$	$S_{k,j}$
2	$S^2_{2k,l+2}$	$S^2_{2k,l+1}$	$S^2_{2k,l}$
3	$S^3_{4k,g+2}$	$S^3_{4k,g+1}$	$S^3_{4k,g}$
4		...	
...			
i	$S^i_{2^{i-1}k,n+2}$	$S^i_{2^{i-1}k,n+1}$	$S^i_{2^{i-1}k,n}$

Fig. 3. The frame structure

3.4 Adaptive Behavior

As proposed in Section 3.3 our approach has two key factors influencing its behavior, the weight measure to employ and the selection of the θ value. If in the literature, several weight measures, mainly related to classifier accuracy, are available and guarantee a

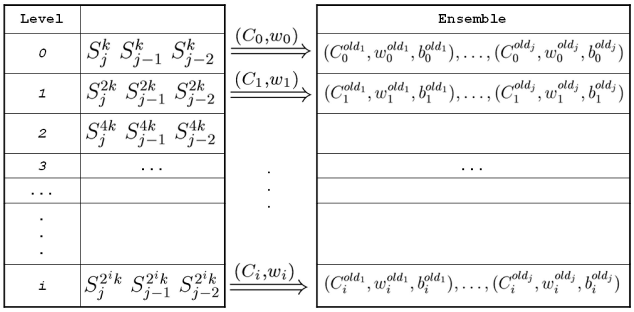

Level
Ensemble
0
1
2
3
i
$S_j^k\ S_{j-1}^k\ S_{j-2}^k$
$S_j^{2k}\ S_{j-1}^{2k}\ S_{j-2}^{2k}$
$S_j^{4k}\ S_{j-1}^{4k}\ S_{j-2}^{4k}$
$S_j^{2^ik}\ S_{j-1}^{2^ik}\ S_{j-2}^{2^ik}$
(C_0,w_0)
(C_1,w_1)
(C_i,w_i)
$(C_0^{old_1},w_0^{old_1},b_0^{old_1}),\ldots,(C_0^{old_j},w_0^{old_j},b_0^{old_j})$
$(C_1^{old_1},w_1^{old_1},b_1^{old_1}),\ldots,(C_1^{old_j},w_1^{old_j},b_1^{old_j})$
$(C_i^{old_1},w_i^{old_1},b_i^{old_1}),\ldots,(C_i^{old_j},w_i^{old_j},b_i^{old_j})$

Fig. 4. The overall system architecture

Activation Threshold Algorithm.

```
1:  if (oneModel() = true) then
2:      θ ← 0.00; activation threshold initialized
3:      oldActModels ← 1
4:      return
5:  end if
6:  actModels ← getActiveModel(θ)
7:  if (actModels > oldActModels) then
8:      θ ← θ + 0.01; increment threshold value
9:  else if (actModels < oldActModels) then
10:     θ ← θ div 2; decrement threshold value
11: end if
12: oldActModels ← actModels
13: return
```

Fig. 5. Pseudo-code of the activation threshold algorithm

good reliability of the system, the θ threshold represents the real key factor for the quality of our approach.

In our experiments [18], we noticed that the reliability of the system is heavily influenced by the θ value. As we shall present in Section 4.3, independently from the data set employed, activation values which are too high (or too small) decrease the predictive power of the ensemble. On the one hand, in case of relatively stable data, small activation threshold values limit the use of large sets of classifiers. On the contrary, large threshold values damage the selective ensemble in the case of concept drifts. In the cited experiments, the θ value was fixed by the user and it did not change through time. Only our experience and the experimental results drove the selection of the right value.

In this work we introduce an adaptive approach for varying the value of the activation threshold through time, thus influencing the overall behavior of the entire system, based on data change reaction. The basic idea of the adaptive method is similar to the additive-increase / multiplicative-decrease algorithm adopted by TCP Internet protocol for managing the transfer rate value used in TCP congestion avoidance.

The pseudo-code of the method for managing the activation threshold is proposed in Figure 5. The algorithm is quite simple. When the first model is inserted in the structure, the activation threshold and the number of active models are initialized (Steps 1-5). Successively, every time a new model is inserted in the ensemble, the procedure at Step 6 computes how many models will be activated with the current θ value. If the number of models potentially activatable is higher than the old one (Step 7), the threshold is increased. This situation happens when the data distribution remains stable and the new inserted model is immediately enabled (Step 7-8). Increasing the threshold value, we can obtain a better exploitation of the ensemble. On the contrary, if the number of active models decreases from the previous invocation, the threshold has to be decreased. It is useless and dangerous to maintain the current value, since a data change might be in progress (Step 9-10). It is worth observing that, if the number of models does not change between the two invocations, the threshold does not change, since there is no evidence of model improving or data change.

From a computational point of view the algorithm does not introduce appreciable overhead. Only the *getActiveModel()* procedure requires to access the ensemble structure. If we consider n as the number of classifiers storable in the ensemble, the complexity of the algorithm is linear in $O(n)$.

The experimental section demonstrates that our system is no more heavily influenced by θ value, since it changes automatically, adapting it to data distribution.

4 Comparative Experimental Evaluation

4.1 Data Sets

Several synthetic data sets and a real one were introduced in our experiments. This kind of data enables an exhaustive investigation about the reliability of the systems involving different scenarios. The data behavior can be described exactly, characterizing the number of concept drifts, the rate between a change to another and the number of irrelevant attributes, or the percentage of noisy data.

`LED24`: Proposed by Breiman et al. in [6], this generator creates data for a display with 7 LEDs. In addition to the 7 necessary attributes, 17 irrelevant boolean attributes with random values are added, and 10 percent of noise is introduced, to make the solution of the problem harder. This type of data generates only stable data sets.

`Stagger`: Introduced by Schlimmer and Granger in [28], this problem consists of three attributes, namely `colour` $\in \{green, blue, red\}$, `shape` $\in \{triangle, circle, rectangle\}$, and `size` $\in \{small, medium, large\}$, and a class `y` $\in \{0,1\}$. In its original formulation, the training set includes 120 instances and consists of three target concepts occurring every 40 instances. The first set of data is labeled according to the concept `color` $= red \wedge$ `size` $= small$, while the others include `color` $= green \vee$ `shape` $= circle$ and `size` $= medium \vee$ `size` $= large$. For each training instance, a test set of 100 elements is randomly generated according to the current concept.

`cHyper`: Introduced in [7], a data set is generated by using a n-dimensional unit hypercube, and an example x is a vector of n-dimensions $x_i \in [0,1]$. The class boundary is a hyper-sphere of radius r and center c. Concept drifting is simulated by changing the c position by a value Δ in a random direction. This data set generator

Table 1. Description of data sets

dataSet	#inst	#attrs	#irrAttrs	#classes	%noise	#drifts
$\mathtt{LED24}_{a/b/test}$	10k / 100k / 25k	24	17	10	10%	none
$\mathtt{Hyper}_{a/b/test}$	10k / 100k / 25k	15	0	2	0	none
$\mathtt{Stagger}_{a/test}$	1200 / 120k	9	0	2	0	3 (every 400) / (every 40k)
$\mathtt{Stagger}_{b/test}$	12k / 1200k	9	0	2	0	3 (every 4k) / (every 400k)
$\mathtt{cHyper}_{a/b/c}$	10k / 100k / 1000k	15	0	2	10%	20 (every 500) / (every 5k) / (every 50k)
$\mathtt{cHyper}_{test}$	250k	15	0	2	10%	20 (every 12.5k)
Cyclic	600k	25	0	2	5%	15×4 (every 10k)
$\mathtt{Cyclic}_{test}$	150k	25	0	2	5%	15×4 (every 2.5k)

introduces noise by randomly flipping the label of a tuple with a given probability. Two additional data sets, namely Hyper and Cyclic are generated using this approach. Hyper does not consider any drifts, while Cyclic proposes the problem of periodic recurring concepts.

KddCup99: this real data set concerns the significant problem of automatic and real-time detection of cyber attacks [31]. The data includes a series of network connections collected from two weeks of LAN network traffic. Each record can either correspond to a normal connection or an intrusive one. Each connection is represented by 42 attributes (34 numerical), such as the *duration* of the connection, the number of *bytes transmitted*, and the *type of protocol* used, e.g. tcp, udp. The data contains 23 training attack types, that can be further aggregated into four categories, namely *DOS*, *R2L*,*U2R*, and *Probing*. Due to its instable nature, KddCup99 is largely employed to evaluate several data streams classification systems, including [3,16]

The features of the data sets actually employed are reported in Table 1. The stable LED24 and Hyper are useful for testing whether the mechanism for change reaction has implications for the reliability of the systems. The evolving data sets test different features of a stream classification system. The Stagger problem verifies, if all the systems can cope with concept drift, without considering any problem dimensionality. Then, the problem of learning in the presence of concept drifting is evaluated with the other data sets, also considering a huge quantity of data with cHyper.

4.2 Systems

Different popular stream ensemble methods are introduced in our experiments. All the systems expect the data streams to be divided into chunks based on a well-defined value. All the approaches are implemented in Java 1.6 with MOA [32] and WEKA libraries [33] for the implementation of the basic learners and employ complete non-approximate data for the mining task.

Fix: This approach is the simplest one. It considers a fixed set of classifiers, managed as a FIFO queue. Every classifier is unconditionally inserted in the ensemble, removing the oldest one, when the ensemble is full.

SEA: A complete description and evaluation of this system can be found in [30]. In this case classifiers are not deleted indiscriminately. Their management is based on a weight measure related to model reliability. This method represents a special case of our selective ensemble, where only one level is defined.

DWM: This system is introduced in [24,25]. The approach implemented here considers a set of data as input to the algorithm, and a batch classifier as the basic one. A weight management is introduced, but differently from SEA, every classifier has a weight associated with it, when it is created. Every time the classifier makes a mistake, its weight decreases.

Oza: This system implements the online bagging method of Oza and Russell [26] with the addition of the `ADWIN` technique [5] as a change detector and as estimator of the weight of the boosting method.

Single: This approach employs an incremental single model with EDDM [13,4] techniques for drift detection. Both Oza and Single were tested using ASHoeffdingTree and naïve Bayes models available in MOA.

4.3 Results

All the experiments were run on a PC with Intel E8200 DualCore with 4Gb of RAM, employing Linux Fedora 10 (kernel 2.6.27) as operating system. Our experiments consider a frame with 8 levels of capacity 3. Every high-order snapshot is built by adding 2 snapshots. This frame size is large enough to consider snapshots that represent big portions of data at higher-levels. For each level, an ensemble of 8 classifiers was used. The tests were conducted comparing the use of the naïve Bayes (NB), and the decision tree (DT) as base classifiers. In all the cases, we compare our Selective Ensemble (SE) (with fixed model activation threshold set to 0.1 and 0.25) with our Adaptive Selection Ensemble ASE. For each data generator, a collection of 100 training sets (and corresponding test sets) are randomly generated with respect to the features outlined in Table 1. Every system is run, and the average accuracy and 95% of interval confidence are reported. Each test consists of a set of 100 observations. All the statistics reported are computed according to the results obtained.

Results with Stable Data Sets. The results obtained with stable data sets confirm that the drift detection approach provided by each system does not heavily influence its overall accuracy. With `LED24` and `Hyper` problems, all the systems reach a quite accurate result. Table 2 reports the results obtained with `Hyper` data sets using the naïve Bayes approach. These results can be compared with the ones provided in Table 3 in Section 4.3, where the concept drifting problem is added to the same type of data.

It is worth observing that there are no significant differences between the results obtained by SE approach, varying the model activation threshold. The new ASE approach provides a result in line with the best ones. The adaptive behavior mechanism does not negatively influence the reliability of the system in the case of stable data streams. On the contrary, the new approach enables a better ensemble exploitation.

Moreover, Table 2 highlights that Single model requires a large quantity of data to provide a good performance. Finally, Fix_{64} and SEA_{64} provide good results that, compared with the ones obtained by the same systems analyzing the `cHyper` and `Cyclic`

Table 2. Results using naïve Bayes with the Hyper problem

	Hyper$_a$ / Hyper$_b$ *avg*	*std dev*	*conf*
ASE	92.74 / 93.93	1.92 / 2.88	0.38 / 0.49
SE$_{0.1}$	92.72 / 93.92	2.20 / 2.52	0.43 / 0.50
SE$_{0.25}$	92.70 / 93.91	2.22 / 2.53	0.43 / 0.50
Fix$_{64}$	91.82 / 92.35	2.54 / 2.98	0.50 / 0.58
SEA$_{64}$	92.97 / 94.44	1.80 / 1.64	0.50 / 0.32
DWM$_{64}$	91.82 / 92.76	2.12 / 2.74	0.42 / 0.54
Oza$_{64}$	92.39 / 93.73	2.30 / 0.40	0.45 / 0.08
Single	90.04 / 92.68	3.25 / 2.82	0.64 / 0.55

problems, demonstrate that these kinds of approaches guarantee appreciable results only with a quite stable phenomenon. They do not provide a fast reaction to concept drift, since the number of models involved in the classification task is constant in time, and when a drift occurs, they have to change a large part of the models, before classifying new concepts correctly.

Table 3. Overall results with the cHyper problem

	cHyper$_a$ / cHyper$_b$ / cHyper$_c$ - decision tree *avg*	*std dev*	*conf*
ASE	83.58 / 88.72 / 93.19	0.51 / 0.40 / 0.28	0.10 / 0.08 / 0.06
SE$_{0.1}$	84.05 / 89.43 / 93.09	0.49 / 0.40 / 0.32	0.10 / 0.08 / 0.06
SE$_{0.25}$	78.42 / 86.10 / 91.86	0.86 / 0.35 / 0.23	0.17 / 0.07 / 0.23
Fix$_{64}$	70.26 / 82.02 / 90.62	2.58 / 1.23 / 0.13	0.51 / 0.24 / 0.13
SEA$_{64}$	70.26 / 82.14 / 90.04	2.58 / 1.10 / 0.14	0.51 / 0.22 / 0.14
DWM$_{64}$	77.75 / 85.18 / 92.65	1.94 / 0.60 / 0.14	0.38 / 0.04 / 0.14
Oza$_{64}$	81.99 / 89.60 / 92.40	0.97 / 0.37 / 0.25	0.19 / 0.07 / 0.25
Single	81.50 / 87.85 / 89.99	1.60 / 0.70 / 0.34	0.31 / 0.14 / 0.34

	cHyper$_a$ / cHyper$_b$ / cHyper$_c$ - naïve Bayes *avg*	*std dev*	*conf*
ASE	87.52 / 92.23 / 95.94	0.38 / 0.43 / 0.33	0.09 / 0.08 / 0.06
SE$_{0.1}$	87.62 / 92.62 / 95.98	0.42 / 0.43 / 0.47	0.08 / 0.09 / 0.09
SE$_{0.25}$	79.90 / 86.80 / 92.14	0.83 / 0.40 / 0.22	0.16 / 0.08 / 0.22
Fix$_{64}$	73.72 / 83.69 / 94.16	2.60 / 1.35 / 0.40	0.51 / 0.26 / 0.40
SEA$_{64}$	73.72 / 84.23 / 94.78	2.60 / 1.27 / 0.31	0.51 / 0.25 / 0.31
DWM$_{64}$	85.93 / 92.18 / 95.63	1.76 / 0.18 / 0.38	0.35 / 0.04 / 0.38
Oza$_{64}$	80.01 / 87.31 / 89.78	1.23 / 0.54 / 0.56	0.24 / 0.11 / 0.56
Single	81.25 / 89.47 / 93.34	2.02 / 0.87 / 0.84	0.40 / 0.17 / 0.84

Results with Evolving Data Sets. Table 3 reports the overall results obtained analyzing the cHyper problem, considering both decision tree and naïve Bayes models. Differently from the results obtained with stable data sets, the active model threshold influences the overall results. Varying the value from 0.1 to 0.25, and especially considering cHyper$_a$ and cHyper$_b$, SE system presents a difference even larger than 6% between the two values. On the contrary, our ASE approach provides an accuracy in line with the best one, even considering standard deviation. This demostrates that, without knowing the ideal threshold value for model activation, our ASE approach represents the right solution to the different situations involved in a stream scenario, and simulated by the three cases of the `cHyper` problem. As stated in the previous section, it is worth observing the poor performances of Fix and SEA in the case of evolving data. These obsevations are further validated by the results obtained with the `Stagger` problem, that essentially follow the ones proposed in Table 3.

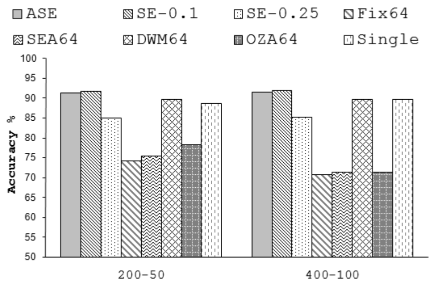
Finally, Table 4 outlines the resources required by the systems. The memory requirements were tested using NetBeans 6.8 Profiler. We can state that Single requires less memory than ensemble methods, which need a quantity of memory that is essentially linear with respect to the number of classifiers stored in the ensemble. The different nature of the two classes of systems influences this value. The average memory required by our system is slightly higher than the others, since our system manages two different structures, as suggested at the end of Section 3.3. The run time behavior confirms this trend. In this case the drift detection approach influences the execution time of a method. Let us compare the bagging method Oza with respect to DWM, SEA_{64} and ASE. These tests highlight that incremental single model systems are faster than ensemble ones, since they have to update only one model. On the contrary, considering the accuracy, single model systems rarely provide best average values. Finally, Oza guarantees an appreciable reliability with every data set, but its execution time is definitely higher than the others.

Table 4. $\texttt{cHyper}_c$ time and memory required

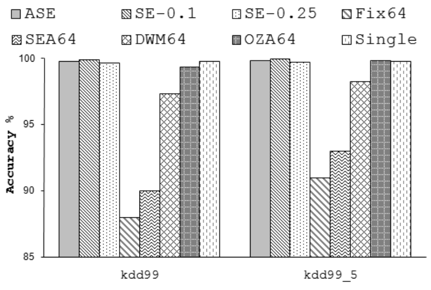
	decision tree		naïve Bayes	
	avg used heap (KB)	*run time (sec.)*	*avg used heap (KB)*	*run time (sec.)*
ASE	9276	82.40	7572	27.42
SE	9233	80.80	7894	27.45
Fix_{64}	8507	47.54	5317	23.82
SEA_{64}	7980	152.07	5371	97.76
DWM_{64}	5111	77.56	5137	21.21
Oza_{64}	10047	393.93	6664	290.24
Single	5683	11.54	5399	8.26

Figure 6a shows the results obtained considering the `Cyclic` problem. The latter are presented considering the naïve Bayes approach and analyzing different rates between the chunk size and the elements to classify. As shown in Figure 6a, even in this case, our ASE approach is in-line with the $\text{SE}_{0.1}$ and better than the others. Since this problem presents recurring concepts, our approach can exploit the selective ensemble better than the others, since some models which are currently out of context are not deleted by the system, but simply disabled. If a concept becomes newly valid, the model can be reactivated. This behavior is still valid, even in the case of the adaptive approach.

We conclude this section, proposing the results obtained analyzing the `KddCup99` problem, and considering the decision tree approach. In this case, only an execution is run considering the whole data set. As shown is Figure 6b, the approaches employing an advanced method to keep track of concept drift propose an accuracy in line with the ones obtained by Aggarwal et al. in [3]. Even in this case, ASE proposes a performance comparable with $\text{SE}_{0.1}$, showing that the adaptive behavior guarantees a good level of reliability. The run time requirements needed for analysing `KddCup99` dataset are in line with the ones proposed in Table 4.



(a) `Cyclic` and naïve Bayes model.



(b) `KddCup99` and decision tree model.

Fig. 6. Average accuracy with `Cyclic` ans `KddCup99` problems

5 Conclusions

Starting from the requirements constrained by the unpredictable nature of streaming data, this paper proposed an adaptive selective ensemble approach for data streams classification. The aim of this work is represented by a new adaptive behavior for an ensemble model selection approach. The new feature enables the system to automatically adapt the active model threshold to the current stream status. The idea is not providing a fixed value of the threshold set up experimentally, but letting its value automatically adapt to the data flow changes. When data are quite stable, the system can use a large part of the ensemble. On the contrary, when data changes the threshold, it has to be reduced to disable the not up-to-date models. The preliminary results show that, with respect to the use of a fixed threshold, our adaptive algorithm provides a slightly worse performance than the ones using the best value of the threshold. Unfortunately, the choice of the best value is not always feasible, and if a wrong selection is made, the system loses its precision. Our adaptive approach does not require any assumption about active model values and displays good adaptation to the different scenarios. This work represents a first step to guarantee a system completely adaptable to the different streaming factors. As future works, our aims are to test our adaptive model in a real stream application with real data. Moreover, we are currently studying the introduction of runtime monitoring tools for automatically adapting our system, e.g varying the number of frame levels, or the models available for each layer, dynamically considering memory consumption and time response constraints.

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: Proceedings of the 2003 International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, pp. 81–92 (2003)
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: A framework for projected clustering of high dimensional data streams. In: Proceedings of the 2004 International Conference on Very Large Data Bases (VLDB 2004), Toronto, Canada, pp. 852–863 (2004)

3. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: On demand classification of data streams. In: Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (KDD 2004), Seattle, WA, pp. 503–508 (2004)
4. Baena-Garcia, M., del Campo-Avila, J., Fidalgo, R., Bifet, A., Ravalda, R., Morales-Bueno, R.: Early drift detection method. In: International Workshop on Knowledge Discovery from Data Streams (2006)
5. Bifet, A., Holmes, G., Pfahringer, B., Kirby, R., Gavaldá, R.: New ensemble methods for evolving data streams. In: Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining, pp. 139–148 (2009)
6. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth International Group, Belmont (1984)
7. Chu, F., Zaniolo, C.: Fast and Light Boosting for Adaptive Mining of Data Streams. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 282–292. Springer, Heidelberg (2004)
8. Cohen, L., Avrahami, G., Last, M., Kandel, A.: Info-fuzzy algorithms for mining dynamic data streams. Applied Soft Computing 8(4), 1283–1294 (2008)
9. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD 2000), Boston, MA, pp. 71–80 (2000)
10. Domingos, P., Hulten, G.: A general method for scaling up machine learning algorithms and its application to clustering. In: Proceedings of the 18th International Conference on Machine Learning (ICML 2001), Williamstown, MA, pp. 106–113 (2001)
11. Folino, G., Pizzuti, C., Spezzano, G.: Mining Distributed Evolving Data Streams using Fractal GP Ensembles. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 160–169. Springer, Heidelberg (2007)
12. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining data streams: a review. ACM SIGMOD Records 34(2), 18–26 (2005)
13. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: SBIA Brazilian Symposium on Artificial Intelligence, pp. 286–295 (2004)
14. Gama, J., Pinto, C.: Discretization from data streams: applications to histograms and data mining. In: Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 2006), Dijon, France, pp. 662–667 (2006)
15. Gama, J., Fernandes, R., Rocha, R.: Decision trees for mining data streams. Intelligent Data Analysis 10(1), 23–45 (2006)
16. Gao, J., Fan, W., Han, J., Yu, P.S.: On appropriate assumptions to mine data streams: Analysis and practice. In: Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), Omaha, NE, pp. 143–152 (2007)
17. Gilbert, A., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Fast, small-space algorithms for approximate histogram maintenance. In: Proceedings of the 2002 Annual ACM Symposium on Theory of Computing (STOC 2002), Montreal, Quebec, Canada, pp. 389–398 (2002)
18. Grossi, V.: A New Framework for Data Streams Classification. Ph.D. thesis, Supervisor Prof. Franco Turini, University of Pisa (2009), `http://etd.adm.unipi.it/theses/available/etd-11242009-124601/`
19. Grossi, V., Turini, F.: Stream mining: a novel architecture for ensemble based classification. Accepted as full paper by Internl. Journ. of Knowl. and Inform. Sys., forthcoming, draft (2011), `www.di.unipi.it/~vgrossi`
20. Grossi, V., Turini, F.: A new selective ensemble approach for data streams classification. In: Proceedings of the 2010 International Conference in Artificial Intelligence and Applications (AIA 2010), Innsbruck, Austria, pp. 339–346 (2010)

21. Guha, S., Koudas, N., Shim, K.: Data-streams and histograms. In: Proceedings of the 2001 Annual ACM Symposium on Theory of Computing (STOC 2001), Heraklion, Crete, Greece, pp. 471–475 (2001)
22. Hulten, G., Spencer, L., Domingos, P.: Mining time changing data streams. In: Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD 2001), San Francisco, CA, pp. 97–106 (2001)
23. Klinkenberg, R.: Learning drifting concepts: Example selection vs. example weighting. Intelligent Data Analysis 8, 281–300 (2004)
24. Kolter, J.Z., Maloof, M.A.: Using additive expert ensembles to cope with concept drift. In: Proceedings of the 22nd International Conference on Machine learning (ICML 2005), Bonn, Germany, pp. 449–456 (2005)
25. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: An ensemble method for drifting concepts. Journal of Machine Learning Research 8, 2755–2790 (2007)
26. Oza, N.C., Russell, S.: Online bagging and boosting. In: Proceedings of 8th International Workshop on Artificial Intelligence and Statistics (AISTATS 2001), Key West, FL, pp. 105–112 (2001)
27. Pfahringer, B., Holmes, G., Kirkby, R.: Handling Numeric Attributes in Hoeffding Trees. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 296–307. Springer, Heidelberg (2008)
28. Schlimmer, J.C., Granger, R.H.: Beyond incremental processing: Tracking concept drift. In: Proceedings of the 5th National Conference on Artificial Intelligence, Menlo Park, CA, pp. 502–507 (1986)
29. Scholz, M., Klinkenberg, R.: An ensemble classifier for drifting concepts. In: Proceeding of 2nd International Workshop on Knowledge Discovery from Data Streams, in Conjunction with ECML-PKDD 2005, Porto, Portugal, pp. 53–64 (2005)
30. Street, W.N., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. In: Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD 2001), San Francisco, CA, pp. 377–382 (2001)
31. The UCI KDD: University of California: KDD Cup 1999 Data, `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`
32. The University of Waikato: MOA: Massive Online Analysis (August 2009), `http://www.cs.waikato.ac.nz/ml/moa`
33. The University of Waikato: Weka 3: Data Mining Software in Java, Version 3.6, `http://www.cs.waikato.ac.nz/ml/weka`
34. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining (KDD 2003), Washington, DC, pp. 226–235 (2003)