Kernel-Based Selective Ensemble Learning for Streams of Trees

Valerio Grossi, Alessandro Sperduti
Department of Pure and Applied Matematics
University of Padova
{grossi, sperduti}@math.unipd.it

Abstract

Learning from streaming data represents an important and challenging task. Maintaining an accurate model, while the stream goes by, requires a smart way for tracking data changes through time, originating concept drift. One way to treat this kind of problem is to resort to ensemble-based techniques. In this context, the advent of new technologies related to web and ubiquitous services call for the need of new learning approaches able to deal with structured-complex information, such as trees. Kernel methods enable the modeling of structured data in learning algorithms, however they are computationally demanding. The contribute of this work is to show how an effective ensemble-based approach can be deviced for streams of trees by optimizing the kernel-based model representation. Both efficacy and efficiency of the proposed approach are assessed for different models by using data sets exhibiting different levels and types of concept drift.

1 Introduction

The advent of new technologies mainly related to web and ubiquitous services has determined the generation, at a rapid rate, of massive unbounded sequences of data elements, i.e. data streams. Data streams need to be processed quickly, using bounded memory resources, and keeping output errors as small as possible. Former approaches involved both unstructured (see, for example [Gaber *et al.*, 2005]) and (semi-)structured data elements (e.g., [Asai *et al.*, 2002; Kudo and Matsumoto, 2004; Bifet and Gavaldà, 2008; Bifet and Gavaldà, 2009]). These approaches, however, assumed the input distribution of data to be time-invariant. In many cases, however, this is hardly the case, since data streams tend to evolve with time, e.g. think of a stream of news or of stock market indices. This phenomenon is referred to as *concept drift*. By adopting the formalization presented in [Klinkenberg, 2004], if we model a stream as an infinite set of elements $e_1, \ldots, e_j, \ldots$, a data stream can be divided into batches, $b_1, b_2, \ldots, b_n, \ldots$, where $b_i \equiv \ e_{i_1}, \ldots, e_{i_{n_i}}$. For each batch b_i, data is independently distributed w.r.t. a distribution $P_i(e)$. Depending on the amount and type of concept drift, $P_i(e)$ will differ from $P_{i+1}(e)$. In the context of data streams classification involving unstructured data elements, two main class of solutions have been proposed: instance selection and ensemble learning. Examples of the first method are Very Fast Decision Trees (VFDT) [Domingos and Hulten, 2000] and its improvements for concept drifting reaction and numerical attributes managing [Hulten *et al.*, 2001; Gama and Pinto, 2006; Pfahringer *et al.*, 2008; Cohen *et al.*, 2008]. Ensemble learning employs multiple classifiers, extracting new models from scratch and deleting the out-of-date ones continuously. Online approaches for bagging and boosting have been proposed in [Oza and Russell, 2001; Chu and Zaniolo, 2004; Bifet *et al.*, 2009]. Alternative methods are described in [Street and Kim, 2001; Wang *et al.*, 2003; Scholz and Klinkenberg, 2005; Kolter and Maloof, 2007; Grossi and Turini, 2011a], where an ensemble of weighted-classifiers is employed to cope with concept drifting. An interesting comparison between different techniques not only in terms of accuracy, but also including computational features, such as memory and time required by each system, is presented in [Bifet *et al.*, 2009].

While the approaches cited above are more or less effective in dealing with concept drifting, all of them introduce an additional computational burden that may be the major reason for which, up to now, no approach for dealing with concept drift in streams of (semi-)structured data elements has been proposed. In fact, methods devised for this type of data are already computationally intensive and a naïve application of the above approaches to them soon becomes computationally infeasible. This is a pity, since the number of applications involving (semi-)structured data is constantly growing. For example, the data exchanged between different web services are codified by XML files that take the form of tree-typed documents; the automated managing of blogs, news and emails coupled with the high interaction between users create new challenges in natural language processing; in this context, all the sentences are typically represented by trees enriched with semantic information to be processed by a machine.

In this paper, we propose a new kernel-based ensemble method, based on the architecture defined in [Grossi and Turini, 2011b] and the on-line kernel-based approach for stream of trees presented in [Aiolli *et al.*, 2006]. We demonstrate how a well-known approach can be employed and reinvented to solve concept drifting problems in tree-structured learning. Starting from the problem of classifying trees, we

employ a kernel-based approach for defining an ensemble of classifiers, that not only improves the overall accuracy of the base approach but guarantees a fast computation. In this context, a variant of the Perceptron algorithm called DAG-Perceptron is employed to define a new approach for learning trees even in high-dynamical environments. The prototyped system is completely evaluated and compared in terms of accuracy and computational time with the online single DAG-Perceptron. Our results demonstrated that the different models extracted from different levels of time granularity, and dynamically selected using current data distribution, provide a better result than a single flat model continuously updated.

2 DAG-Perceptron for Tree streams

The DAG-Perceptron [Aiolli *et al.*, 2006; 2007] has been defined to deal with classification tasks involving a stream of trees. It is an efficient version of an on-line kernel-perceptron algorithm, adapted to tree-kernels, able to speed-up training of a factor above 3. From a functional point of view, this corresponds to keep in memory the set of the already seen trees for which the perceptron prediction was erroneous. More specifically, let $\mathcal{T}$ be a stream of example pairs (T_i, y_i), where T_i are trees, and $y_i \in \{-1, +1\}$. Then, at iteration $t+1$, the scoring function of the perceptron for a new tree T is defined by

$$S_{t+1}(T) = \sum_{i=1}^{t} \alpha_i K(T_i, T), \qquad (1)$$

where $\alpha_i \in \{-1, 0, +1\}$ is 0 whenever $sign(S_i(T_i)) = y_i$, and y_i otherwise; $K(\cdot,\cdot)$ is a tree kernel, e.g. the SubSet Tree kernel (SST). The SST kernel counts the number of matching subset trees between two input trees. A subset tree of a tree T is a structure which is rooted in a node of T and it satisfies the constraint that each of its nodes is connected to either all its children in T or none of them. The SST kernel is defined as $K(T_1, T_2) = \sum_{v_1 \in V_{T_1}} \sum_{v_2 \in V_{T_2}} C(v_1, v_2)$, where V_{T_1} and V_{T_2} are the sets of vertices of trees T_1 and T_2, respectively, $C(v_1, v_2)$ is recursively computed according to the following rules: *i*) if $label(v_1) \neq label(v_2)$ then $C(v_1, v_2) = 0$; *ii*) if $label(v_1) = label(v_2)$ and v_1 is a preterminal then $C(v_1, v_2) = 1$; *iii*) if $label(v_1) = label(v_2)$ and v_1 is not a preterminal then $C(v_1, v_2) = \prod_{j=1}^{nc(v_1)}(1 + C(ch_j[v_1], ch_j[v_2]))$, where $nc(v_1)$ is the number of children of v_1 and $ch_j[v]$ is the j-th child of vertex v. For a detailed description of the SST kernel see [Collins and Duffy, 2002]. By disregarding trees that have null weight, eq. (1) can be rewritten as

$$S_{t+1}(T) = \sum_{(T_i, y_i) \in M_t} y_i K(T_i, T)$$

where $M_t = \{(T_i, y_i) \in \mathcal{T} : i \leq t,\ S_i(T_i) \neq y_i\}$ is the *model* at time t of the perceptron. It is trivial to show that the cardinality of M, and consequently the memory required for its storage, grows up with the number of tree presentations. However, M_t can be efficiently maintained in a compact structure represented by a Directed Acyclic Graph (DAG) where any (sub)tree s_i, occurring f^+_{t,s_i} times in trees with $y_i = +1$ and f^-_{t,s_i} times in trees with $y_i = -1$, is represented only once and its root v_{s_i} is annotated with $f_{t,v_{s_i}} = f^+_{t,s_i} - f^-_{t,s_i}$. In [Aiolli *et al.*, 2006] it is shown that this compact representation can be exploited to efficiently compute the score for a tree by computing each $C(\cdot,\cdot)$ entry only once:

$$S_t(T) = \sum_{v_i \in DAG_{M_t}} \sum_{v_k \in T} f_{t,v_i} C(v_i, v_k)$$

The above computation can easily be adapted to a voted version of the perceptron, thus improving the generalization ability of the system while still maintaining an efficient computation. This can be obtained by allocating for each vertex v of the DAG an additional variable, used for computing the score of the voted perceptron, which maintains the cumulated sum $f^{voted}_{t,v} = \sum_{i \leq t : sign(S_i(T_i)) \neq y_i} (t - i + 1) \cdot f_{i,v}$, where we assume $f_{i,v} = 0$ for all vertices v that first appear in the DAG after time i.

3 Selective Dag-Perceptron Ensemble

Capturing concept drifting is one of the main challenges in the context of mining data streams. A system should be conceived for ensuring a good trade-off between data reduction, and a powerful representation of all the evolving features.

Data reduction is needed because building a prediction model that embeds all the information concerning erroneously classified instances coming from a stream of data runs into the risk of generating a model of unbounded size. Thus, the mining process needs to select and/or compress information about misclassified instances. On the other side, a powerful representation of all the evolving features is required to cope with concept drifting, so that, as soon as a new concept appears, its characteristic features are identified and properly represented for prediction purposes. Characteristic features of out-of-date concepts still need to be preserved, since there is the possibility that those concepts will appear again in the future. One serious downside of allowing a powerful representation of all the evolving features is the need for a potentially huge amount of memory, i.e. it goes against the data reduction requirement we described above.

The approach proposed in [Grossi and Turini, 2011b] for ensuring a good trade-off between these two requirements has been to resort to a compact-layered ensemble method implementing a multi-resolution data reduction strategy. The adopted strategy can be summarized as follows. Streaming data, as long as it arrives, is partitioned in chunks, $ck_1, ck_2, \ldots$. Recent chunks are used to build models belonging to the ensemble at level 1. These models, $M_1, M_2, \ldots$ are supposed to capture up-to-date concepts occurring in $ck_1, ck_2, \ldots$. This is the way how the system is able to identify and to retain (as long as memory availability allows it) evolving features. The number of these models, let us say k, is a structural feature of the ensemble and it is decided apriori. When model M_{k+1} is created, the eldest model M_1 is discarded from level 1, so to make space for M_{k+1}. The discarded model M_1 is inserted at level 2, and as soon as another model is discarded from level 1, i.e. M_2, due to the insertion at level 1 of the most recently created model M_{k+2}, M_1 is

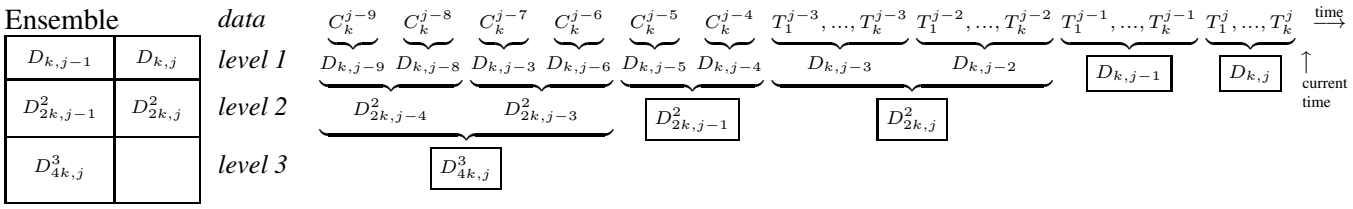


Figure 1: DAGs and their order. Only framed DAGs belong to the ensemble; the others contributed to create high-order DAGs.

aggregated with M_2 at level 2, and it will not be aggregable anymore for that level. The resulting aggregated model M_{1+2} is an high-order model since it covers, in a compressed form, all the instances belonging to chunks ck_1 and ck_2. This process is repeated till the number of models at level 2 exceeds k, in which case the outcoming model from level 2 is inserted into level 3. This is repeated until the maximum number of levels (again, defined apriori) is reached, in which case the outgoing model is lost. It is now clear that a model belonging at level i summarizes data coming from 2^{i-1} chunks, thus allowing a progressive aggregation of information along with the level index. Let us now explain how this strategy can be applied to a stream of trees.

Our approach, called EnDAG, is based on the structure explained before to build an ensemble. Such ensemble constitutes a tool for concept drift reaction, guaranteeing a fast kernel computation. The latter is one of the key aspects of this approach, since in complex structures as trees, computing kernel scores is computationally expensive.

Each tree-chunk generates a DAG-Perceptron model. DAGs are employed to maximize the number of elements for training classifiers. Generally, a model mined from a small set of elements tends to be less accurate than the one extracted from a large data set. This assumption is true in "traditional" learning contexts, but in a stream environment it is not necessarily satisfied. Due to concept drifting, large data set can include out-of-date concepts.

To clarify the role of each DAG inside our system, we distinguish two types of models; the DAGs computed directly from the stream, and the ones obtained by the aggregation of existing DAGs (high-order DAGs). The creation of a high-order DAG does not require any further access to the original data. Fig. 1 highlights the relation between the two types of models, showing an example of our structure called *frame*. A frame is defined by three values indicating the number of levels, the capacity of each level, and the number of DAGs that can be aggregated in a high-order DAG. For each tree-chunk of size k, $C_k^j = \{T_1^j, ..., T_k^j\}$, a DAG-Perceptron ($D_{j,k}$) is computed. Successively, when the stream evolves the models are further aggregated in high-order DAGs, e.g. $D^2_{j,2k}$ or $D^3_{j,4k}$. For each level, only the highlighted models are actually stored in the structure. The higher is the level, the larger is the data chunk considered by the high-order models. Moreover, Fig. 1 shows the logical naïve management of the frame structure. All the DAGs computed directly from the stream are inserted at level 1 and represent the basic element of the structure. Since each level has a finite capacity, when a level is full the out-coming DAG is not deleted but it is inserted (or aggregated) into the successive level. DAGs coming out from the last level are simply deleted. From a logical perspective, this structure can be viewed as a set of sliding windows representing different complementary portions of the stream, with the properties that only the basic models access original data.

This logical view cannot be implemented directly. Managing all DAGs separately, as show in Fig. 1, tends to repeat the same structures in different DAGs with a waste of resources, in terms of time and memory. For this reason, the ensemble is efficiently represented by a single DAG (EnDAG) where *a frame structure is associated with each node*, thus avoiding to replicate the same structure several times and enabling a fast score computation for all needed models. The EnDAG is created and subsequently updated by an operation that adds a DAG to it:

$$DAG_j \oplus EnDAG \equiv ADD(\overrightarrow{EnDAG}, DAG_j) \quad (2)$$

As defined in eq. (2), the insertion of DAG_j in EnDAG requires two distinct operations, namely *shifting* and *adding*.

Shitfing: Algorithm 1 defines the $\overrightarrow{EnDAG}$ operation. Since each level has a finite capacity, before adding a new element to the ensemble, we have to set a slot as free for each vertex of EnDAG. Given a vertex v, the value of the last slot of level i is forwarded to the first slot of level $i+1$ and all the remaining values of level i are shifted to the right.[1] The forwarded value is summed to the first slot of level $i+1$ (aggregation) if a value not already aggregated is present, otherwise it is inserted into the first slot of level $i+1$ after that the same procedure described above has be recursively applied to level $i+1$. The recursive invocation is repeated until a value is aggregated or the last level of the frame is reached.

Adding: when a slot is set as free, the operation of adding a new DAG in the ensemble can be performed. We recall that our ensemble is represented by a single DAG, where for each vertex we manage a frame structure. The adding operation involves two different situations:

1. $\forall v_i \in DAG_j$ such that $\exists v_k \in EnDAG$ for which $tree(v_i) \equiv tree(v_k)$, then $v_k.frame[1,1] \leftarrow f_{v_i}$. The function $tree(v)$ returns the tree rooted in v.
2. $\forall v_i \in DAG_j$ such that $\nexists v_k \in EnDAG$ for which $tree(v_i) \equiv tree(v_k)$, then: *i)* vertex v_i is added[2] to

[1] This is actually implemented by a circular array.

[2] Vertices of DAG_j are examined following an inverted topolog-

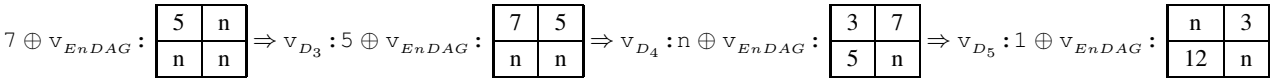


Figure 2: Example of application of the $\oplus$ operator to vertex $v_{EnDAG} \in EnDAG$ and matching vertices $v_{D_j},\ j = 2, \ldots, 5$ (i.e., $tree(v_{EnDAG}) \equiv tree(v_{D_j})$). The *null* value is represented by the n character.

Algorithm 1 The $\overrightarrow{EnDAG}$ algorithm

```
1: for all v ∈ EnDAG do
2:    insert(v, 2, v.frame[1, limit]);
3:    for j = limit − 1 to 1 do
4:       v.frame[1, j + 1] ← v.frame[1, j];
5:    end for
6:    v.frame[1, 1] ← null;
7: end for
```

```
insert(v, lev, val)
  if v.aggregable[lev] = 1 then
    v.frame[lev, 1] ← v.frame[lev, 1] + val;
    v.aggregable[lev] ← 0;
  else
    insert(v, lev + 1, v.frame[lev, limit]);
    for j = limit − 1 to 1 do
      v.frame[lev, j + 1] ← v.frame[lev, j];
    end for
    v.frame[lev, 1] ← val;
    v.aggregable[lev] ← 1;
  end if
```

EnDAG; *ii)* $v_k.frame[1,1] \leftarrow f_{v_i}$ and all the remaining slots of $v_k.frame[,]$ get the $null$ value.

Fig. 2 shows, for a given vertex $v \in EnDAG$, an example of evolution of the content of $v.frame$ (of size 2×2) following the insertion into the ensemble of DAGs generated after reading some tree-chunks. For each insertion, next to the ':' character, both the value associated with the input vertex v_{D_i} ($i = 2, \ldots, 5$) and the current content of $v.frame$, as well as the result of the $\oplus$ operation, are shown.

Fig. 3 provides a complete picture about the insertion of a new DAG into EnDAG. For all the vertices already available in the ensemble, we only update the frame structure. Otherwise, as in the case of vertex with label q, a new vertex is added to EnDAG with a frame that contains only a value (the f_v value) in first position.

This construction, where all the common substructures are not duplicated between DAGs, enables an efficient high-order DAGs creation and a fast score computation. In fact, given a tree, we can compute the score associated with every DAG in the structure by visiting EnDAG only once, and simply considering the values different from *null* inside each frame.

It is worth to notice that the computation of high-order DAGs does not imply any loss of information as long as any high-order DAG is lost:

Theorem: Let $D = \{D_1, \ldots, D_m\}$ be a set of DAGs (models). Given a tree T and $S_D(T) = \sum_{D_i \in D} S_{D_i}(T)$

$$\begin{aligned} S_D(T) &= S_{EnDAG}(T) \\ &= \sum_{v \in EnDAG, u \in T} C(v,u) \sum_{i,j}^{r,c} v.frame[i,j], \end{aligned}$$

where $EnDAG = D_1 \oplus \cdots \oplus D_m$, and EnDAG uses a vertex frame of size $r \times c$ such that $\left(c \sum_{i=1}^{r} 2^{i-1}\right) = c(2^r - 1) \geq m$.

Proof: the proof of this theorem is derived from the one presented in [Aiolli *et al.*, 2006] and not reported here due to space limitations.

Finally, it is worth observing that a data stream involves an infinite set of elements, that cannot be stored completely even with this structure. In fact, the actual size of the frame is decided beforehand on the basis of the available computational resources.

3.1 Ensemble management

The actual ensemble management is a four-phase approach. ***i.*** For each tree-chunk C_i, a triple (D_i,w_i,b_i) representing a model, its weight and a boolean value b_i is extracted from C_i (see below for more info about w_i and b_i). ***ii.*** Subsequently, $D_i \oplus EnDAG$ is computed; notice that each entry of any EnDAG vertex's frame will correspond to a model. ***iii***. Since data distribution can change through time, at time t the DAGs currently in the structure (i.e., the corresponding entries into the frames associated with vertices of EnDAG) are re-weighed as follows: $\forall i,\ w_i = accuracy(D_i, C_t^{0.25})$ where $accuracy(D_i, C_t^{0.25})$ is the accuracy of the model D_i computed over 25% of trees belonging to tree-chunk C_t; it should be noticed that w_i is independent from which specific $v \in EnDAG$ is considered. ***iv.*** A set of *active DAGs* (i.e., frame entries) is selected, setting the boolean value b_i associated with a D_i as *true*. The set of *active models* is selected

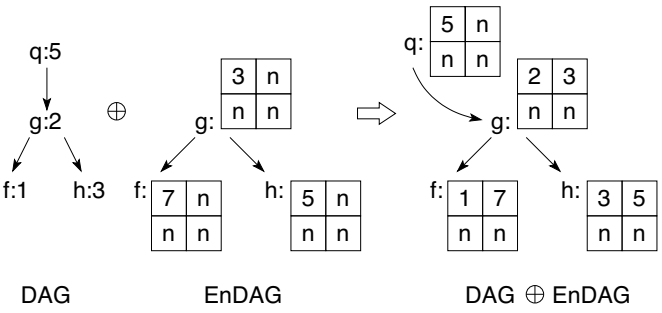


Figure 3: Example of update of EnDAG. $\forall v \in DAG$ we report $label(v) : f_v$, while $\forall u \in EnDAG$ we report $label(u) : u.frame$. The $null$ value is represented by n.

ical order. This guarantees that when a vertex v must be inserted into EnDAG, all its children already belong to EnDAG.

	Num. trees	Num. vertices avg / min / max	Depth avg / min / max	Maximum out-degree avg / min / max
Set_a	1000	15,85 / 5 / 107	7,75 / 2 / 24	2,30 / 2 / 8
Set_b	229814	15,57 / 5 / 167	7.65 / 2 / 28	2,28 / 2 / 14
Set_c	1500000	15,72 / 5 / 256	7.74 / 2 / 41	2,29 / 2 / 15

Table 1: Original data statistics.

based on the value of an activation threshold θ. All the classifiers that differ at most θ from the best classifier with the highest weight are enabled. This behavior allows the ensemble to dynamically select those models that best match the concepts currently represented in the stream. However, currently disabled models are not deleted from the ensemble since they can be enabled in the future in case associated matching concepts come back as relevant.

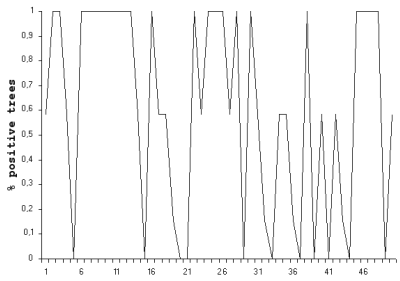
4 Comparative Experimental Evaluation

We compared 3 applications of our approach with the incremental single flat model (labelled `Dag`) proposed in [Aiolli *et al.*, 2007; 2006]. We considered a weighed-ensemble approach, labeled `WE`, where all the models are active and the final score is computed, even considering the weight associated with each DAG. In the remaining two approaches, the selective behavior is enabled. The final class is computed considering the weighed score or a majority class of the active DAGs only. These two approaches are labeled `SE` and `SEM` respectively. In all the cases, even the voted approaches are tested as well.

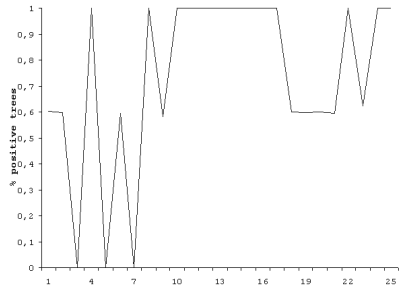
The aim of the faced task is to predict, for each tree, the average vertex out-degree. All the experiments involve the use of semi-real data, where it is possible to control concept drift, and test how the latter influences the final results. Original data includes two sets of trees computed as Part-of-Speech of sentences taken from PropBank dataset. Specifically, the input trees are real, taken from the POS tagging, while the task is artificially created (by randomly changing the threshold through time) for studying the behavior of the systems in the presence of different drifting phenomena. Table 1 reports the statistics about the datasets.

This approach is motivated by the fact that, before assessing our approach with a real-world task, we must demonstrate the reliably of the approach on controlled tasks. Due to space limitation, only results on "hard" concept drifts are presented. It is worth observing that in case of small concept fluctuations, an ensemble of classifiers is by construction (due to the average operator) robust.

Since our aim is to analyze the reaction of the systems to concept drifting, three families of tests were considered. All the systems were stressed by considering different situations, where concept drift may occur. Given a set of trees, a drift is simulated by classifying the trees based on their average out-degree value. In particular, a threshold h is randomly changed through time to assign the class value to a tree. If a tree T has an average vertex out-degree higher than h, T is classified as +1, otherwise t class is -1. Figure 4 highlights the concept drifting phenomenon inside a specific train data of `DS-0` and `DS-2`. With reference to Table 2, `DS-0` in-



(a) `DS-0`



(b) `DS-2`

Figure 4: Concept drift phenomenon. Percentage of positive trees (y-axis) through time (x-axis).

volves the use of exactly 1000 trees taken from Set_a, namely 500 for the test set and 500 for the train set. Every 500 elements a drift may occur. This situation represents a pure concept drifting phenomenon, since from a drift to the next one the same set of trees changes its class value. `DS-1` shares the same philosophy of `DS-0`, but a change may occur every 1000 trees and the same tree can be randomly repeated many times both in the train and the test set. `DS-1` proposes a real context, where some features appear in multiple and unpredictable forms, while others are more unlikely. Finally, `DS-2` selects a subgroup from Set_b, and estimates the reliability of each system by varying the ratio between a drift and the next one. The described datasets are freely available at `http://www.math.unipd.it/~sperduti/TreeCD`.

All the systems were implemented in Java 1.6 and the experiments were run on a PC with Intel E8200 DualCore with 4Gb of RAM, employing Windows Vista as OS.

Our experiments consider frames of size 8×4. This frame size is large enough to consider DAGs representing big portions of data at higher-levels. The dimension of the frame is selected to avoid loss of information, so to have a fair comparison of our approach versus the single flat DAG model, which does not lose information.

The model activation threshold θ is set to 0.1 as the default value used in [Grossi and Turini, 2011b]. For each test, a collection of 10 training sets (and corresponding test sets of equal size as the training sets) are randomly generated with respect to the features outlined in Table 2. Every system is run, and the accuracy is computed by using a train-and-test approach. After a learning batch of 200 trees, the reliability

	Original data set train	Num. trees train	Avg. different trees trees	Avg. shared concept drifts	Avg. effective
DS-$0_{a/b/c}$	Set_a	25k / 50k / 100k	500 - 500	50 / 100 / 200	47,2 / 97,5 / 190.7
DS-$1_{a/b/c}$	Set_a	25k / 50k / 100k	358 - 575	8893.2 / 17738.4 / 35626.4	20.1 / 40.1 / 77.7
DS-$2_{a/b/c}$	Set_b	25k / 50k / 75k	24729.6 / 49153.2 / 73340.3	4162.8 / 15183.9 / 31754.2	20.2 / 18.9 / 18.1

Table 2: Description of datasets composition and statistics.

is tested with a test set of 200 unseen elements. Finally, the average accuracy and 95% of interval confidence is reported.

Table 3 presents the overall accuracy reported by the systems. As a general observation, we can state that our SE and Dag approaches provide an average accuracy higher than the other ones even considering the confidence value. We additionally performed a test of significance between these two systems, highlighting when the difference between SE and Dag is statistically relevant. Moreover, Table 3 shows how the selective behavior is important to guarantee a powerful tool for reacting to concept drift. In this situation, WE approach does not guarantee a reliable performance in the presence of multiple data changes. DS-2 data set shows how the accuracy of SE and Dag tends to be the same, increasing the period between two changes. The use of the voted version does not provide any improvement in the accuracy of our approaches. This is due to the nature of the voted approach that implicitly simulates an ensemble method.

The run-time behavior of the systems is proposed in Table 4. The latter highlights how our SE (and SEM) guarantees a performance that outperforms all the others. This result is provided by the structure of EnDAG that manages all the DAGs in the ensemble in a unique structure avoiding the repetition of substructures. Tests not reported here for lack of space showed that the described EnDAG implementation, on the average, improves speed of a factor 4 on the naïve approach that maintains all the DAGs separated.

These results show that, in the presence of concept drifting, in most cases the accuracy of our system is increased with respect to a single model. Moreover, if we consider SE and Dag approaches, they demonstrate that even if in some cases the accuracy of the two systems is similar, the run time values guarantee that our SE approach is more suitable to be employed in a streaming environment.

As a final test, we studied the behavior of our systems (excluding voting) on a large dataset of 1 million of trees generated from Set_c in Table 1. We tested only our SE with Dag, since the two systems are the most representative ones, given the past tests. In particular, if we consider the behavior proposed by SE, i.e. an average accuracy of 95.72% and runtime of 211976.6 seconds, the results show that Dag is much slower than SE, being able to process only 42.2% of the input trees in the same amount of time needed by SE to complete its task.

5 Conclusions

Kernel methods provide a powerful tool for modeling structured objects in learning algorithms. Unfortunately, they require a high computational complexity to be used in streaming environments.

This work is the first that demonstrates how kernel methods can be employed to define an ensemble approach able to quickly react to concept drifting and guarantees an efficient kernel computation. Our experimental assessment demonstrates that for high dynamical data streams our system provides an high level of reliability both in terms of accuracy and time performance.

Acknowledgments

Valerio Grossi was supported by the FSE fellowship Id: *2105/1/13/2215/2009*.

References

[Aiolli *et al.*, 2006] F. Aiolli, G. Da San Martino, A. Sperduti, and A. Moschitti. Fast on-line kernel learning for trees. *Data Mining, IEEE Internl. Conf. on*, 0:787–791, 2006.

[Aiolli *et al.*, 2007] F. Aiolli, G. Da San Martino, A. Sperduti, and A. Moschitti. Efficient kernel-based learning for trees. In *CIDM 2007*, pages 308 – 315, Honolulu, HI, 2007.

[Asai *et al.*, 2002] T. Asai, H. Arimura, K. Abe, S. Kawasoe, and S. Arikawa. Online algorithms for mining semi-structured data stream. *Data Mining, IEEE Internl. Conf. on*, page 27, 2002.

[Bifet and Gavaldà, 2008] A. Bifet and R. Gavaldà. Mining adaptively frequent closed unlabeled rooted trees in data streams. In *Proceeding of the 14th ACM SIGKDD Internl. Conf. on Knowl. Disc. and data mining*, KDD '08, pages 34–42, New York, NY, USA, 2008. ACM.

[Bifet and Gavaldà, 2009] Albert Bifet and Ricard Gavaldà. Adaptive xml tree classification on evolving data streams. In *ECML/PKDD (1)*, pages 147–162, 2009.

[Bifet *et al.*, 2009] A. Bifet, G. Holmes, B. Pfahringer, R. Kirby, and R. Gavaldá. New ensemble methods for evolving data streams. In *Proc. of the 15th Internl. Conf. on Knowl. Disc. and Data Mining*, pages 139–148, 2009.

[Chu and Zaniolo, 2004] F. Chu and C. Zaniolo. Fast and light boosting for adaptive mining of data streams. In *Proc. of the 8th Pacific-Asia Conf. Advances in Knowl. Disc. and Data Mining (PAKDD'04)*, pages 282–292, Sydney, Australia, 2004.

[Cohen *et al.*, 2008] L. Cohen, G. Avrahami, M. Last, and A. Kandel. Info-fuzzy algorithms for mining dynamic data streams. *App. Soft Comput.*, 8(4):1283–1294, 2008.

[Collins and Duffy, 2002] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: Kernels over

	DS-0		DS-1		DS-2	
	avg	*conf*	*avg*	*conf*	*avg*	*conf*
WE	78,62 / 78,44 / 75,60	1,63 / 1,37 / 1,41	86,09 / 83,26 / 82,69	2,13 / 1,96 / 1,80	87,41 / 89,47 / 90,32	2,21 / 1,34 / 1,87
SE	**88,60 / 89,33 / 88,54**	0,80 / 0,77 / 0,44	94,94 / **94,79 / 95,08**	0,71 / 0,51 / 0,45	93,37 / 93,90 / 95,00	1,23 / 0,84 / 1,01
SEM	86,87 / 87,77 / 86,88	1,10 / 0,95 / 0,50	93,72 / 93,69 / 94,06	0,89 / 0,58 / 0,80	92,03 / 92,58 / 93,99	1,54 / 0,92 / 1,26
Dag	87,66 / 88,01 / 87,42	0,68 / 0,71 / 0,29	94,30 / 94,19 / 94,21	0,71 / 0,42 / 0,32	92,45 / 93,82 / 94,74	0,74 / 0,37 / 0,61
WE_v	80,13 / 79,94 / 77,49	1,98 / 1,39 / 1,72	87,91 / 85,32 / 84,98	2,10 / 2,02 / 2,78	88,33 / 90,25 / 91,22	2,18 / 1,40 / 1,73
SE_v	**89,08 / 89,60 / 88,92**	0,73 / 0,73 / 0,40	**94,94 / 94,74 / 94,98**	0,79 / 0,52 / 0,83	93,36 / 93,76 / 94,84	1,19 / 0,87 / 1,13
SEM_v	87,16 / 88,02 / 87,26	0,87 / 0,92 / 0,50	93,87 / 93,75 / 94,09	0,95 / 0,60 / 0,83	92,39 / 92,69 / 94,00	1,41 / 1,03 / 1,41
Dag_v	87,71 / 87,82 / 87,11	0,61 / 0,73/ 0,30	93,64 / 93,39 / 93,41	0,73 / 0,42 / 0,35	92,23 / 93,70 / 94,69	0,66 / 0,33 / 0,53

Table 3: Accuracy results. Results for $SE_{(v)}$ that are statistically significant with respect to $Dag_{(v)}$ are shown in bold.

	DS-0		DS-1		DS-2	
	avg	*std. dev*	*avg*	*std. dev*	*avg*	*std. dev*
WE	189,8 / 473,2 / 1313,7	30,29 / 95,15 / 178,47	140,5 / 395,9 / 1012,5	41,47 / 67,33 / 153,92	701,3 / 3096,5 / 6726,5	411,14 / 1088,58 / 3138,02
SE	81,7 / 195,8 / 529,8	14,20 / 39,77 / 77,19	65,4 / 184,0 / 461,9	26,99 / 42,45 / 97,88	257,8 / 1228,4 / 2986,7	156,01 / 465,76 / 1815,89
SEM	81,5 / 195,1 / 531,9	14,29 / 39,83 / 77,83	65,2 / 184,8 / 464,4	26,33 / 43,06 / 98,14	258,4 / 1223,8 / 2980,4	156,54 / 464,83 / 1816,10
Dag	165,0 / 476,8 / 1102,3	40,05 / 33,31 / 56,84	77,1 / 225,1 / 622,6	11,19/ 32,46 / 43,72	989,8 / 4054,4 / 8662,2	413,20 / 897,57 / 2676,28
WE_v	192,0 / 477,4 / 1335,5	32,35 / 94,49 / 190,12	141,8 / 399,6 / 1027,9	41,65 / 65,40 / 160,15	706,3 / 3134,3 / 6783,4	413,03 / 1103,95 / 3145,10
SE_v	83,4 / 196,9 / 540,5	14,23 / 38,03 / 70,56	67,2 / 185,9 / 462,6	27,39 / 42,59 / 99,44	256,7 / 1190,6 / 2529,1	155,92 / 407,27 / 1229,89
SEM_v	83,0 / 196,7 / 535,6	14,00 / 38,66 / 68,94	67,4 / 186,5 / 464,7	27,63 / 43,38 / 97,95	256,5 / 1193,8 / 2526,8	156,02 / 408,30 / 1230,55
Dag_v	166,2 / 480,1 / 1110,4	39,85 / 33,93 / 54,91	77,9 / 225,7 / 622,0	11,50 / 31,03 / 43,70	995,0 / 4067,9 / 8720,7	416,87 / 894,77 / 2679,03

Table 4: Average run-time in seconds.

discrete structures, and the voted perceptron. In *ACL02*, 2002.

[Domingos and Hulten, 2000] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. of the 6th Internl. Conf. on Knowl. Disc. and Data Mining (KDD'00)*, pages 71–80, Boston. MA, 2000.

[Gaber *et al.*, 2005] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM SIGMOD Records*, 34(2):18–26, 2005.

[Gama and Pinto, 2006] J. Gama and C. Pinto. Discretization from data streams: applications to histograms and data mining. In *Proc. of the 2006 ACM symposium on Applied computing (SAC'06)*, pages 662–667, Dijon, France, 2006.

[Grossi and Turini, 2011a] V. Grossi and F. Turini. An adaptive selective ensemble for data streams classification. In *In Proc. of the Third Internl. Conf. on Agents and Artificial Intelligence (ICAART 2011)*, pages 136–145, Rome, Italy, 2011.

[Grossi and Turini, 2011b] V. Grossi and F. Turini. Stream mining: a novel architecture for ensemble based classification. *Accepted as full paper by Internl. Journ. of Knowl. and Inform. Sys., DOI: 10.1007/s10115-011-0378-4*, 2011.

[Hulten *et al.*, 2001] G. Hulten, L. Spencer, and P. Domingos. Mining time changing data streams. In *Proc. of the 7th Internl. Conf. on Knowl. Disc. and Data Mining (KDD'01)*, pages 97–106, San Francisco, CA, 2001.

[Klinkenberg, 2004] R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intell. Data Analy.*, 8:281–300, 2004.

[Kolter and Maloof, 2007] J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journ. of Mach. Learn. Res.*, 8:2755–2790, 2007.

[Kudo and Matsumoto, 2004] Taku Kudo and Yuji Matsumoto. A boosting algorithm for classification of semi-structured text. In *EMNLP*, pages 301–308, 2004.

[Oza and Russell, 2001] N. C. Oza and S. Russell. Online bagging and boosting. In *Proc. of 8th Internl. Workshop on Artificial Intelligence and Statistics (AISTATS'01)*, pages 105–112, Key West, FL, 2001.

[Pfahringer *et al.*, 2008] B. Pfahringer, G. Holmes, and R. Kirkby. Handling numeric attributes in hoeffding trees. In *PAKDD'08*, pages 296–307, Osaka, Japan, 2008.

[Scholz and Klinkenberg, 2005] M. Scholz and R. Klinkenberg. An ensemble classifier for drifting concepts. In *Proceeding of 2nd Internl. Workshop on Knowl. Disc. from Data Streams, in conjunction with ECML-PKDD 2005*, pages 53–64, Porto, Portugal, 2005.

[Street and Kim, 2001] W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proc. of the 7th Internl. Conf. on Knowl. Disc. and Data Mining (KDD'01)*, pages 377–382, San Francisco, CA, 2001.

[Wang *et al.*, 2003] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. of the 9th Internl. Conf. on Knowl. Disc. and Data Mining (KDD'03)*, pages 226–235, Washington, DC, 2003.