Accurate Ensembles for Data Streams: Combining Restricted Hoeffding Trees using Stacking

Albert Bifet ABIFET@CS.WAIKATO.AC.NZ
Eibe Frank EIBE@CS.WAIKATO.AC.NZ
Geoffrey Holmes GEOFF@CS.WAIKATO.AC.NZ
Bernhard Pfahringer BERNHARD@CS.WAIKATO.AC.NZ
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Editor: Masashi Sugiyama and Qiang Yang

Abstract

The success of simple methods for classification shows that is is often not necessary to model complex attribute interactions to obtain good classification accuracy on practical problems. In this paper, we propose to exploit this phenomenon in the data stream context by building an ensemble of Hoeffding trees that are each limited to a small subset of attributes. In this way, each tree is restricted to model interactions between attributes in its corresponding subset. Because it is not known *a priori* which attribute subsets are relevant for prediction, we build exhaustive ensembles that consider all possible attribute subsets of a given size. As the resulting Hoeffding trees are not all equally important, we weigh them in a suitable manner to obtain accurate classifications. This is done by combining the log-odds of their probability estimates using sigmoid perceptrons, with one perceptron per class. We propose a mechanism for setting the perceptrons' learning rate using the `ADWIN` change detection method for data streams, and also use `ADWIN` to reset ensemble members (i.e. Hoeffding trees) when they no longer perform well. Our experiments show that the resulting ensemble classifier outperforms bagging for data streams in terms of accuracy when both are used in conjunction with adaptive naive Bayes Hoeffding trees, at the expense of runtime and memory consumption.

Keywords: Data Streams, Decision Trees, Ensemble Methods

1. Introduction

When applying boosting algorithms to build ensembles of decision trees using a standard tree induction algorithm, the tree inducer has access to all the attributes in the data. Thus, each tree can model arbitrarily complex interactions between attributes in principle. However, often the full modeling power of unrestricted decision tree learners is not necessary to yield good accuracy with boosting, and it may even be harmful because it can lead to overfitting. Hence, it is common to apply boosting in conjunction with depth-limited decision trees, where the growth of each tree is restricted to a certain level. In this way, only a limited set of attributes can be used in each tree, but the risk of overfitting is reduced. In the extreme case, only a single split is allowed and the decision trees degenerate to decision

stumps. Despite the fact that each restricted tree can only model interactions between a limited set of attributes, the overall ensemble is often highly accurate.

The method presented in this paper is based on this observation. We present an algorithm that produces a classification model based on an ensemble of restricted decision trees, where each tree is built from a distinct subset of the attributes. The overall model is formed by combining the log-odds of the predicted class probabilities of these trees using sigmoid perceptrons, with one perceptron per class. In contrast to the standard boosting approach, which forms an ensemble classifier in a greedy fashion, building each tree in sequence and assigning corresponding weights as a by-product, our method generates each tree in parallel and combines them using perceptron classifiers by adopting the stacking approach (Wolpert, 1992). Because we are working in a data stream scenario, Hoeffding trees (Domingos and Hulten, 2000) are used as the ensemble members. They, as well as the perceptrons, can be trained incrementally, and we also show how `ADWIN`-based change detection (Bifet and Gavaldà, 2007) can be used to apply the method to evolving data streams.

There is existing work on boosting for data streams (Oza and Russell, 2001b), but the algorithm has been found to yield inferior accuracy compared to bagging (Bifet et al., 2009b,a). Moreover, it is unclear how a boosting algorithm can be adapted to data streams that evolve over time: because bagging generates models independently, a model can be replaced when it is no longer accurate, but the sequential nature of boosting prohibits this simple and elegant solution. Because our method generates models independently, we can apply this simple strategy, and we show that it yields more accurate classifications than Online Bagging (Oza and Russell, 2001b) on the real-world and artificial data streams that we consider in our experiments.

Our method is also related to work on building ensembles using random subspace models (Ho, 1998). In the random subspace approach, each model in an ensemble is trained based on a randomly chosen subset of attributes. The models' predictions are then combined in an unweighted fashion. Because of this latter property, the number of attributes available to each model must necessarily be quite large in general, so that each individual model is powerful enough to yield accurate classifications. In contrast, in our method, we exhaustively consider all subsets of a given, small, size, build a tree for each subset of attributes, and then combine their predictions using stacking. In random forests (Breiman, 2001), the random subspace approach is applied locally at each node of a decision tree in a bagged ensemble. Random forests have been applied to data streams, but did not yield substantial improvements on bagging in this scenario (Bifet et al., 2010b).

The paper is structured as follows. The next section presents the new ensemble learner we propose. Section 3 details the set-up for our experiments. Section 4 presents the experimental comparison to bagging. In Section 5 we consider ensemble pruning and evaluate the effect of using stacking rather than simple averaging to combine the trees' predictions. Section 6 concludes the paper.

2. Combining Restricted Hoeffding Trees using Stacking

The basic method we present in this paper is very simple: enumerate all attribute subsets of a given user-specified size k, learn a Hoeffding tree from each subset based on the incoming data stream, gather the trees' predictions for each incoming instance, and use these predic-

tions to train simple perceptron classifiers. The question that needs to be addressed is how exactly to prepare the "meta" level data for the perceptrons and what shape it should take.

Rather than using discrete classifications to build the meta level model in stacking, it is common to use class probability estimates instead because they provide more information due to the fact that they represent the degree of confidence that each model has in its predictions. Hence, we also adopt this approach in our method: the meta-level data is formed by collecting the class probability estimates for an incoming new instance, obtained from the Hoeffding trees built from the data observed previously.

The meta level combiner we use is based on simple perceptrons with sigmoid activation functions, trained using stochastic gradient descent to minimize the squared loss with respect to the actual observed class labels in the data stream. We train one perceptron per class value and use the Hoeffding trees' class probability estimates for the corresponding class value to form the input data for each perceptron.

There is one caveat. Because the sigmoid activation function is used in the perceptrons, we do not use the raw class probability estimates as the input values for training them. Rather, we use the log-odds of these probability estimates instead. Let $\hat{p}(c_{ij}|\vec{x})$ be the probability estimate for class i and instance $\vec{x}$ obtained from Hoeffding tree j in the ensemble. Then we use

$$a_{ij} = \log(\hat{p}(c_{ij}|\vec{x})/(1 - \hat{p}(c_{ij}|\vec{x}))$$

as the value of input attribute j for the perceptron associated with class value i.

Let $\vec{a_i}$ be the vector of log-odds for class i. The output of the sigmoid perceptron for class i is then $f(\vec{a_i}) = 1/(1 + e^{-(\vec{w_i}\vec{a_i}+b_i)})$, based on per-class coefficients $\vec{w_i}$ and b_i. We use the log-odds as the inputs for the perceptron because application of the sigmoid function presupposes a linear relationship between $\log(f(\vec{a_i})/(1 - f(\vec{a_i})))$ and $\vec{a_i}$.

To avoid the zero-frequency problem, we slightly modify the probability estimates obtained from a Hoeffding tree by adding a small constant ϵ to the probability for each class, and then renormalize. In our experiments, we use $\epsilon = 0.001$, but smaller values lead to very similar results.

The perceptrons are trained using stochastic gradient descent: the weight vector is updated each time a new training instance is obtained from the data stream. Once the class probability estimates for that instance have been obtained from the ensemble of Hoeffding trees, the input data for the perceptrons can be formed, and the gradient descent update rule can be used to perform the update. The weight values are initialized to the reciprocal of the size of the ensemble, so that the perceptrons give equal weight to each ensemble member initially.

A crucial aspect of stochastic gradient descent is an appropriately chosen learning rate, which determines the magnitude of the update. If it is chosen too large, there is a risk that the learning process will not converge. A common strategy is to decrease it as the amount of training data increases. Let n be the number of training instances seen so far in the data stream, and let m be the number of attributes. We set the learning rate α based on the following equation:

$$\alpha = \frac{2}{2 + m + n}.$$

However, there is a problem with this approach for setting the learning rate in the context we consider here: it assumes that the training data is identically distributed. This

is not actually the case in our scenario because the training data for the perceptrons is derived from the probability estimates obtained from the Hoeffding trees, and these change over time, generally becoming more accurate. Setting the learning rate based on the above equation means that the perceptrons will adapt too slowly once the initial data in the data stream has been processed.

There is a solution to this problem: the stream of predictions from the Hoeffding trees can be viewed as an *evolving* data stream (regardless of whether the underlying data stream forming the training data for the Hoeffding trees is actually evolving) and we can use an existing change detection method for evolving data streams to detect when the learning rate should be reset to a larger value. We do this very simply by setting the value of n to zero when change has been detected. To detect change, we use the `ADWIN` change detector (Bifet and Gavaldà, 2007), which is discussed in more detail in the next section. It detects when the accuracy of a classifier increases or decreases significantly as a data stream is processed. We apply it to monitor the accuracy of each Hoeffding tree in the ensemble. When accuracy changes significantly for one of the trees, the learning rate is reset by setting n to zero. The value of n is then incremented for each new instance in the stream until a new change is detected. This has the effect that the learning rate will be kept relatively large while the learning curve for the Hoeffding trees has a significant upward trend.

2.1 `ADWIN`-based Change Detection

The `ADWIN` change detector comes with nice theoretical guarantees. In addition to using it to reset the learning rate when necessary, we also use it to make our ensemble classifier applicable to evolving data stream, where the original data stream, used to train the Hoeffding trees, changes over time.

`ADWIN` maintains a window of observations and 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, 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.

The strategy we use to cope with evolving data streams using `ADWIN` is very simple, and based on the approach that has been used to make Online Bagging applicable to an evolving data stream in (Bifet et al., 2009b). The idea is to replace ensemble members when they start to perform poorly. To implement this, we use `ADWIN` to detect when the accuracy of one of the Hoeffding trees in the ensemble has dropped significantly. To do this, we can use the same `ADWIN` change detectors that are also applied to detect when the learning rate needs to be reset. When one of the change detectors associated with a particular tree reports a significant drop in accuracy, the tree is reset and the coefficients in the perceptrons that are associated with this tree (one per class value) are set to zero. A new tree is then generated from new data in the data stream, so that the ensemble can adapt to changes.

Note that all trees for which a significant drop is detected are replaced. Note also that a change detection event automatically triggers a reset of the learning rate, which is important because the perceptrons need to adapt to the changed ensemble of trees.

2.2 A Note on Computational Complexity

Our approach is based on generating trees for all possible attribute subsets of size k. If there are m attributes in total, there are $\binom{m}{k}$ of these subsets. Clearly, only moderate values of k, or values of k that are very close to m, are feasible. When k is one, there is no penalty with respect to computational complexity compared to building a single Hoeffding tree, because then there is only one tree per attribute, and an unrestricted tree also scales linearly in the number of attributes. If k is two, there is an extra factor of m in the computational complexity compared to building a single unrestricted Hoeffding tree, i.e. overall effort becomes quadratic in the number of attributes.

$k = 2$ is very practical even for datasets with a relatively large number of attributes, although certainly not for very high-dimensional data (for which linear classifiers are usually sufficient anyway). Larger values of k are only practical for small numbers of attributes, unless k is very close to m (e.g. $k = m - 1$). We have used $k = 4$ for datasets with 10 attributes in our experiments, with very acceptable runtimes. It is important to keep in mind that many practical classification problems appear to exhibit only very low-dimensional interactions, which means small values of k are sufficient to yield high accuracy.

3. Experiment Setup

We have performed several experiments to investigate the performance and resources needed by our new method, and to compare to bagged Hoeffding trees. To measure use of resources, we use time and memory, and also RAM-Hours, a new evaluation measure introduced in (Bifet et al., 2010c), where every RAM-Hour equals a GB of RAM deployed for 1 hour.

We performed our experiments using the new **M**assive **O**nline **A**nalysis (MOA) framework (Bifet et al., 2010a), a data mining software environment for implementing algorithms and running experiments for evolving data streams. We implemented all algorithms in Java within the MOA framework. The experiments were performed on 2.66 GHz Core 2 Duo E6750 machines with 4 GB of memory.

In the rest of this section we briefly describe the datasets and methods used (excluding our new method, which was described in the previous section).

3.1 Real-World Data

We consider two types of data streams, synthetic ones and ones corresponding to real-world problems. To evaluate our method on real-world data, we use three different datasets: two large datasets from the UCI repository of machine learning databases (Frank and Asuncion, 2010), namely Forest Covertype and Poker-Hand, and one other large dataset that has previously been used to study data stream methods, namely the Electricity data (Harries, 1999; Gama et al., 2004). In our experiments with these datasets all numeric attributes have been normalized to the $[0, 1]$ range.

Forest Covertype This data contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The data has $581,012$ instances and 54 attributes. There are 7 classes. The data is often used to evaluate methods for data stream classification, e.g. in (Gama et al., 2003; Oza and Russell, 2001a).

Poker-Hand This data contains $1,000,000$ instances. Each instance in this dataset is an example of a hand consisting of five playing cards drawn from a standard deck of 52 cards. There are 10 predictor attributes in this data because each card is described using two attributes (suit and rank). The class attribute has 10 possible values and describes the poker hand, e.g. full house. The cards are not ordered in the data and a hand can be represented by any permutation, which makes it very difficult to learn for propositional learners such as the ones that are commonly used for data stream classification. Hence, we use a modified version, in which cards are sorted by rank and suit and duplicates have been removed (Bifet et al., 2009b). The resulting dataset contains 829,201 instances.

Electricity The electricity data is described in (Harries, 1999) and has been used for data stream classification in (Gama et al., 2004). It originates from the New South Wales Electricity Market in Australia. In this market, prices are not fixed but set every five minutes, as determined by supply and demand. The data comprises 45,312 instances and 8 predictor attributes. The class label is determined based on the change of price relative to a moving average of the last 24 hours, yielding 2 class values.

In the data stream scenario, it is important to consider evaluation under concept drift. We can use the method from (Bifet et al., 2009b) in conjunction with the above three datasets to create a data stream exhibiting concept drift. To this end, we assume the underlying distributions are fixed and then combine these "pure" distributions by modeling concept drift events as weighted combinations of two distributions. As originally suggested in (Bifet et al., 2009b), the sigmoid function is used to define the probability that an instance is sampled from the "old" stream—the one in place before the concept drift event—or the "new" stream. In this way, two distributions are mixed using a soft threshold function. The exact specification of the operator used to combine two data streams is given in the following definition from (Bifet et al., 2009b).

Definition 1 *Given two data streams a, b, we define $c = a \oplus_{t_0}^{W} b$ as the data stream built by joining the two data streams a and b, where t_0 is the point of change, W is the length of change,* $\Pr[c(t) = b(t)] = 1/(1 + \mathrm{e}^{-4(t-t_0)/W})$ *and* $\Pr[c(t) = a(t)] = 1 - \Pr[c(t) = b(t)]$.

This operator can be applied repeatedly to introduce multiple concept change events, thus making it possible to combine multiple data streams into a single evolving data stream. Different parameters can be used for each change event. For example, the following expression corresponds to the combination of four data streams a, b, c, and d, joined with three parametrized change events: $(((a \oplus_{t_0}^{W_0} b) \oplus_{t_1}^{W_1} c) \oplus_{t_2}^{W_2} d) \ldots$.

We use this operator to join the above three real-world datasets into a single data stream that exhibits concept drift. More specifically, we define a new data stream

$$\text{CovPokElec} = (\text{CoverType} \oplus_{581,012}^{5,000} \text{Poker-Hand}) \oplus_{829,201}^{5,000} \text{ELEC}$$

Note that, to perform the operation, we concatenate the lists of attributes from the three datasets, and the number of classes is set to the maximum number of classes amongst the three datasets.

3.2 Synthetic Data Streams

There is a shortage of publicly available large real-world datasets that are suitable for the evaluation of data stream methods. Thus we also consider synthetic data in our experiments. To avoid bias, we use synthetic data generators that are commonly found in the literature. Most of these data generators exhibit a mechanism that implements concept drift. For each generator, we use 1 million instances in our experiments.

Rotating Hyperplane This data generator, introduced in (Hulten et al., 2001), provides an elegant way of generating time-changing data for the evaluation of data stream classification. The data is generated based on hyperplanes whose orientation and position is varied smoothly over time. Classes are assigned to data points based on which side of a hyperplane they are located. The generator has a parameter that controls the speed of change. We use 10 predictor attributes for this data and 2 class values.

Random RBF Generator This data generator, from (Kirkby, 2007), is also able to produce time-changing data. Like the hyperplane generator, it produces data that is difficult to model accurately using decision tree learners. It first generates a fixed number of radial basis functions with randomly chosen centroids and assigns weights and class labels to them. Data is then generated based on these basis functions, taking their relative weights into account. Concept drift is introduced by moving the centroids with constant speed, as controlled by a parameter. We use several variants of this data, varying the numbers of centroids and the speed of change, but keeping the number of predictor attributes constant at 10 and the number of class values fixed at 5.

LED Generator This generator is one of the oldest data generators that can be found in the literature. It was introduced in (Breiman et al., 1984) and a C implementation can be found in the UCI repository of machine learning databases (Frank and Asuncion, 2010). The task is to classify the 10 digits based on boolean attributes that correspond to the LEDs of a seven-segment LED display. Each attribute value has a 10% chance of being inverted, so the data includes noise. The variant of the data generator that we use additionally also generates 17 completely irrelevant attributes. Concept drift is controlled by a drift parameter that determines the number of attributes with drift.

Waveform Generator This source of synthetic data was also introduced in the CART book (Breiman et al., 1984) and a C implementation is available from the UCI repository. The data has three classes that correspond to three different waveforms. Each waveform is based on a combination of two or three base waves. As in the LED generator, there is a drift parameter determining the number of attributes with drift. The data has 21 attributes.

Random Tree Generator This generator, which is the only generator we use that does not implement concept drift, first constructs a decision tree by randomly choosing attributes to define splits for the internal nodes of the tree. It then assigns randomly chosen class labels to the leaf nodes of this tree. To generate data, it routes uniformly

distributed synthetic instances down the appropriate paths in the tree and assigns class labels based on the corresponding leaf nodes. This generator was introduced in (Domingos and Hulten, 2000) and favours decision tree learners. We use it with 10 attributes and 5 class values.

3.3 Hoeffding Naive Bayes Trees and ADWIN Bagging

Hoeffding trees (Domingos and Hulten, 2000) are state-of-the-art tree inducers in classification for data streams. They exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the goodness of an attribute). Using the Hoeffding bound one can show that the inducer's output is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples.

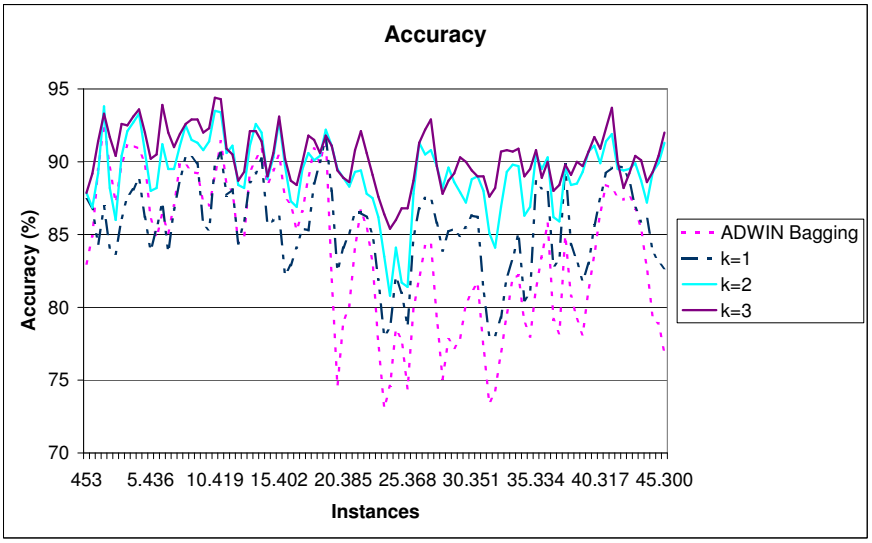
Hoeffding trees perform prediction by choosing the majority class at each leaf. Their predictive accuracy can be increased by adding naive Bayes models at the leaves of the trees. However, (Holmes et al., 2005) identified situations where the naive Bayes method outperforms the standard Hoeffding tree initially but is eventually overtaken. They proposed a *Hoeffding Naive Bayes Tree* (**hnbt**), a hybrid adaptive method that generally outperforms the two original prediction methods for both simple and complex concepts. This method works by performing a naive Bayes prediction per training instance, comparing its prediction with the majority class. Counts are stored to measure how many times the naive Bayes prediction gets the true class correct as compared to the majority class. When performing a prediction on a test instance, the leaf will only return a naive Bayes prediction if it has been more accurate overall than the majority class, otherwise it resorts to a majority class prediction.

Bagging using ADWIN (Bifet et al., 2009b) is based on the online bagging method of (Oza and Russell, 2001b) with the addition of the ADWIN algorithm to detect changes in accuracy for each ensemble member. It yields excellent predictive performance for evolving data streams. When a change is detected, the classifier of the ensemble with the lowest accuracy, as predicted by its ADWIN detector, is removed and a new classifier is added to the ensemble. We use ADWIN Bagging to compare to our new ensemble learner, in both cases using **hnbt** as the base learner for the ensemble members.

4. Comparison with Bagging

Our first experimental evaluation compares our new stacking method using restricted Hoeffding trees with bagging, and we use the datasets discussed in the previous section for this evaluation. The evaluation methodology used was Interleaved Test-Then-Train or Prequential evaluation (based on 10 runs for the artificial data): every example was used for testing the model before using it to train (Gama et al., 2009). This interleaved test followed by train procedure was carried out on the full training set in each case. The parameters of the artificial streams are as follows:

- RBF(x,v): RandomRBF data stream of x centroids moving at speed v.
- HYP(x,v): Hyperplane data stream with x attributes changing at speed v.



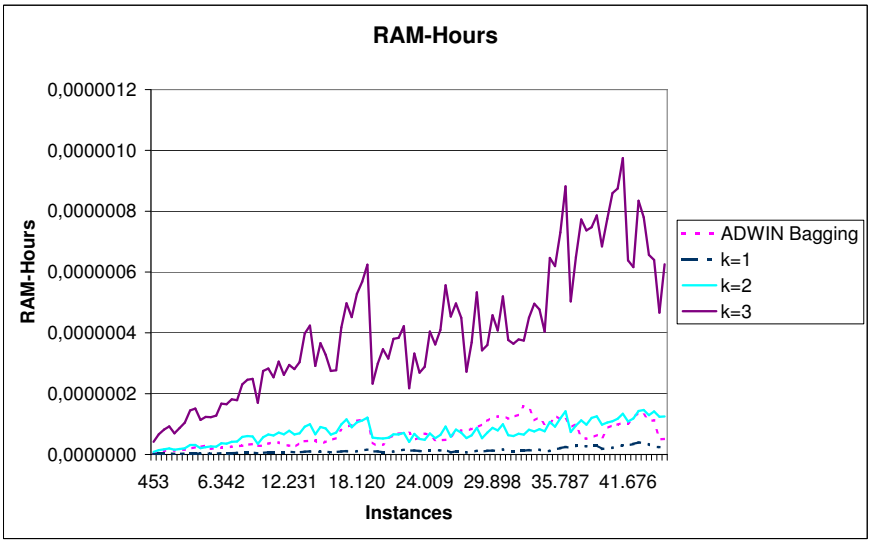
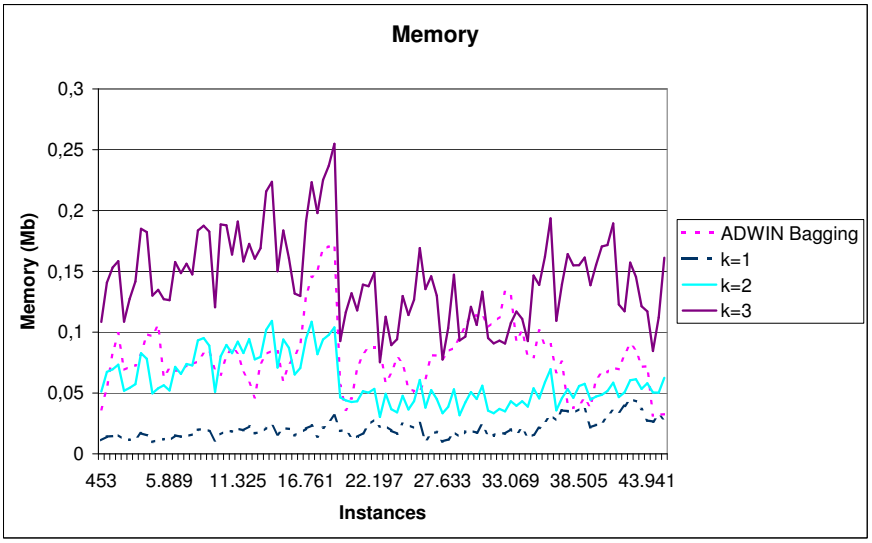


Figure 1: Accuracy and RAM-Hours on the Electricity dataset.



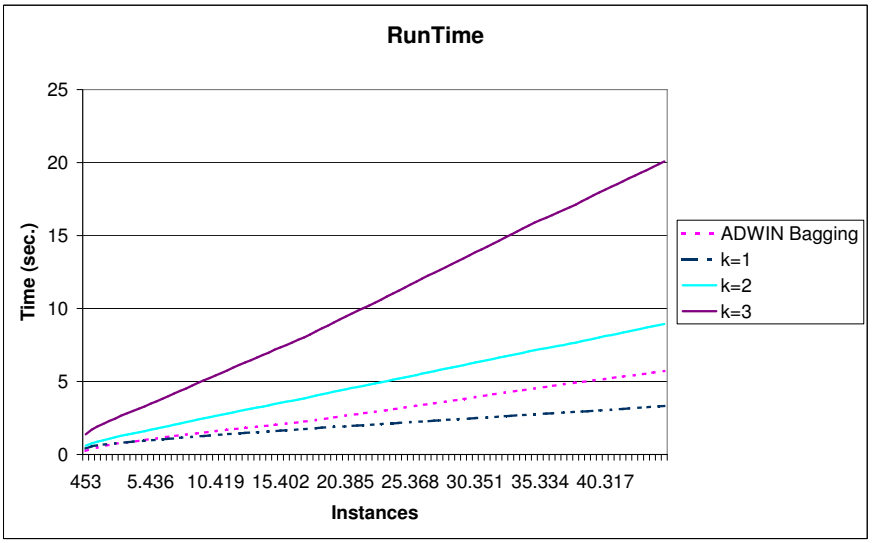


Figure 2: Runtime and memory on the Electricity dataset.

	k=1			k=2			k=4		
	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.	Time
CovType	0.22	88.26	309.53	10.57	**90.64**	11397.01		*ND*	
Electricity	0.05	85.50	2.06	0.14	89.30	6.36	0.43	**90.52**	19.68
Poker-Hand	0.04	76.17	38.31	0.28	79.74	185.95	1.17	**80.67**	1284.38
CovPokElec	0.26	**80.84**	1255.49		*ND*			*ND*	
RBF(50,0)	33.79	52.35 ± 0.23	134.35	5.63	80.57 ± 0.08	492.00	50.27	**87.67** ± 0.04	4041.01
RBF(50,0.001)	18.37	32.18 ± 0.56	100.98	0.14	43.14 ± 0.08	429.24	0.73	**51.19** ± 0.06	3484.85
RBF(10,0.001)	32.78	49.85 ± 0.20	131.64	4.89	74.18 ± 0.07	484.87	37.92	**81.30** ± 0.08	4105.68
RBF(50,0.0001)	0.75	49.20 ± 0.12	62.00	0.21	65.72 ± 0.12	452.44	1.46	**77.44** ± 0.18	3677.13
RBF(10,0.0001)	20.78	50.67 ± 0.16	111.49	2.23	77.27 ± 0.08	476.51	8.05	**85.78** ± 0.04	4102.96
RT	17.95	48.20 ± 0.28	98.34	124.33	66.84 ± 0.93	558.00	141.63	**90.90** ± 0.34	2459.67
LED(50000)	0.50	71.16 ± 0.27	222.10	6.29	72.65 ± 0.09	3620.69	658.99	**72.76** ± 0.06	258630.57
Waveform(50)	51.54	82.00 ± 0.36	243.66	56.64	**85.89** ± 0.21	1732.22		*ND*	
HYP(10,0.001)	5.79	75.91 ± 0.39	61.27	2.33	88.94 ± 0.38	228.39	17.21	**90.93** ± 0.28	1770.61
HYP(10,0.0001)	10.98	75.68 ± 1.32	79.08	5.22	90.34 ± 0.14	257.16	37.23	**91.68** ± 0.14	1950.51
	65.57 Acc. 0.15 RAM-Hours			77.32 Acc. 1.21 RAM-Hours			**81.89** Acc. 73.98 RAM-Hours		

Table 1: Comparison using sets of 1, 2 or 4 attributes. Accuracy is measured as the final percentage of examples correctly classified over the full test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface.

	k=3			ADWIN Bagging 10 Trees			ADWIN Bagging 100 Trees		
	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.	Time
CovType	240.83	**91.46**	241144.76	0.83	84.75	201.59	13.06	84.76	2662.80
Electricity	0.27	**90.33**	13.78	0.17	84.11	3.62	3.98	84.59	30.48
Poker-Hand	0.72	**80.80**	588.20	0.09	74.04	65.27	1.40	71.13	886.78
CovPokElec		*ND*		0.99	**78.36**	425.65	16.90	76.55	6692.23
RBF(50,0)	21.84	85.78 ± 0.07	1834.72	6.11	87.67 ± 0.52	191.84	60.52	**88.48** ± 0.28	2143.47
RBF(50,0.001)	0.38	48.59 ± 0.10	1563.64	0.09	**51.15** ± 0.11	220.50	0.92	44.65 ± 0.04	2332.09
RBF(10,0.001)	17.89	79.13 ± 0.08	1799.88	5.10	80.35 ± 0.25	200.24	56.51	**81.31** ± 0.12	2126.37
RBF(50,0.0001)	0.68	**74.28** ± 0.13	1609.09	0.19	59.94 ± 0.28	225.65	1.90	60.67 ± 0.32	2401.50
RBF(10,0.0001)	5.26	83.58 ± 0.03	1780.45	5.76	83.70 ± 0.24	202.33	67.14	**84.27** ± 0.29	2175.74
RT	249.04	85.06 ± 0.11	1467.27	20.32	91.16 ± 0.15	156.06	202.99	**91.27** ± 0.13	1686.98
LED(50000)	68.97	72.66 ± 0.08	47380.04	1.35	**73.12** ± 0.07	203.19	24.44	73.07 ± 0.06	3044.94
Waveform(50)	199.21	**85.84** ± 0.29	18774.08	1.68	83.63 ± 0.52	277.35	52.33	85.33 ± 0.34	2759.82
HYP(10,0.001)	7.70	**90.40** ± 0.36	788.07	2.59	89.68 ± 0.34	103.86	42.64	89.01 ± 0.46	1192.34
HYP(10,0.0001)	17.70	**91.47** ± 0.15	865.02	6.80	91.04 ± 0.24	117.97	118.73	90.75 ± 0.48	1367.94
	81.49 Acc. 72.00 RAM-Hours			79.48 Acc. 0.04 RAM-Hours			78.99 Acc. 5.67 RAM-Hours		

Table 2: Comparison using sets of 3 attributes and ADWIN Bagging of 10 and 100 trees. Accuracy is measured as the final percentage of examples correctly classified over the full test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface.

- RT: Random Tree data stream
- Waveform(v): Waveform dataset, with v as the value of the drift parameter.
- LED(v): LED dataset, with v as the value of the drift parameter.

We test our method using restricted Hoeffding trees containing 1, 2, 3, and 4 attributes respectively, against Bagging using `ADWIN`. We do not show results for Online Bagging (Oza and Russell, 2001b) as in (Bifet et al., 2009b) it was shown that Bagging using `ADWIN` outperforms Online Bagging. Results for a single Hoeffding Naive Bayes Tree (**hnbt**) are also excluded because Bagging using `ADWIN` (10 trees) is at least two percent more accurate than **hnbt** on all but two of the datasets we used, Poker-Hand and CovPokElec. The former dataset is the only one on which **hnbt** achieves substantially higher accuracy (77.1%) than Bagging using `ADWIN`.

Tables 1 and 2 report the final accuracy, speed, and memory consumption of the classification models induced on the synthetic data and the real-world datasets: Forest CoverType, Poker-Hand, Electricity and CovPokElec (the mixture of the three). Accuracy is measured as the final percentage of examples correctly classified over the test/train interleaved evaluation. Time is measured in seconds, and memory in MB.

As the RandomRBF, Random Tree, and Hyperplane data streams have 10 attributes, the number of attribute sets used by the restricted trees for these datasets are

$$\binom{10}{1} = 10 \quad \binom{10}{2} = 45 \quad \binom{10}{3} = 120 \quad \binom{10}{4} = 210 \quad \binom{10}{5} = 252$$

respectively. However, when the number of attributes is bigger than 15, the number of combinations is huge as combinations grow exponentially. For example, for the Forest CoverType data with 54 attributes we obtain the following figures:

$$\binom{54}{1} = 54 \quad \binom{54}{2} = 1,431 \quad \binom{54}{3} = 24,804 \quad \binom{54}{4} = 316,251$$

We observe that for $k = 2$ ($\binom{m}{2} = m \cdot (m-1)/2$) the number of possible combinations is relatively small, and we can run experiments on all datasets. However, for $k = 4$, the number of combinations is so large ($\binom{m}{4} = m \cdot (m-1) \cdot (m-2) \cdot (m-3)/24$) that for some datasets insufficient memory was available to run the experiments.

The results of our method for different values of k, and the comparison to bagging, show that excellent predictive performance can be obtained on the practical datasets for small values of k: even with $k = 1$, the proposed method outperforms bagging. Further substantial improvements can be obtained by moving from $k = 1$ to $k = 2$. These results indicate that modeling low-dimensional interactions is sufficient for obtaining good performance on the practical datasets we considered.

Also, we note that Bagging using `ADWIN` with 100 trees is not more accurate that using 10 trees. Hence, the good performance of our method is not due to the fact that it uses a larger number of trees. The combination of using attribute subsets and the perceptron weighting methodology is essential for the improvement in accuracy obtained.

The learning curves and model growth curves for the ELEC dataset are plotted in Figures 1 and 2. We use the prequential method (Gama et al., 2009) with a sliding window

of $1,000$ examples to compute the learning curves. We observe that using $k = 1$ we obtain the fastest and most resource-efficient method, but also the worst in accuracy. On the other hand, with $k = 3$ we have the slowest method, the one that uses more RAM-Hours, but also the most accurate. As we increase the complexity of the model using higher values for k, we use more resources, but we improve the method's accuracy. `ADWIN` Bagging appears to use a similar amount of RAM-Hours as with $k = 2$, but its accuracy for this dataset is worse than with $k = 1$.

The results on the artificial data yield a somewhat different picture. On these data streams, larger values of k are necessary in most cases to obtain classification accuracy that is competitive with that of bagging. This is not surprising given the functional relationships underlying the data generation processes. Nevertheless, competitive performance can be obtained with $k = 3$ or $k = 4$.

As we increase the number of attributes used in our method, accuracy improves but the resources needed increase. Obtaining greater accuracy comes at a cost and requires more resources. Depending on the environment that we are working with—sensor networks, handheld computers, or servers—we may need to use different values for k.

5. Pruning and Comparison of Stacking with Simple Averaging

Although the results on the real-world data streams show that small values of k are often sufficient to obtain accurate classifiers, it is instructive to consider whether it is possible, at least in principle, to prune the ensemble by reducing the number of classifiers, without losing accuracy. To investigate this, we performed a further experiment, using only those trees for prediction that exhibited the largest average coefficients in the perceptrons, where the average was taken across class values. Table 3 shows the results for reduced sets of $10, 20$, and 40 classifiers respectively, for $k = 2$. Even for 40 trees, performance is substantially below that of the full ensemble, indicating that a pruning strategy based on simply removing classifiers is not sufficient to maintain high accuracy.

Finally, we investigate whether it is necessary to train a perceptron to combine the trees' predictions. To this end, we show results for simple averaging of the trees' probability estimates in Table 4. We see that using the simple averaging mechanism we use the same resources, but obtain lower accuracy. Hence, combining predictions adaptively is clearly essential. This is due to the fact that not all attribute subsets (and, thus, Hoeffding trees) are equally relevant for prediction.

To conclude our experimental evaluation, let us briefly consider our method in the case where each tree in the ensemble can access a large number of attributes—more specifically, close to the full number of attributes m. This case is interesting to consider because $\binom{m}{k} = \binom{m}{m-k}$, although using $m - k$ as subset size does introduce an extra factor of m in overall time complexity. Table 4 shows results for our method using $m - 2$ and $m - 3$ attributes respectively. Predictive performance on the artificial data streams is very good compared to the low-dimensional cases considered previously (where k was small) because these data streams benefit from modeling higher-dimensional interactions. However, using small values of k is clearly preferable on the real-world data, yielding higher accuracy with lower resource use.

	Top 10 trees			Top 20 trees			Top 40 trees		
	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.	Time
CovType	10.57	52.27	9100.09	10.57	52.66	9073.76	10.57	**54.46**	9052.99
Electricity	0.14	85.44	5.70	0.14	88.08	5.96	0.14	**89.30**	6.36
Poker-Hand	0.28	69.14	158.38	0.28	72.84	167.07	0.28	**75.23**	184.21
CovPokElec	17.89	**55.25**	53622.51		*ND*			*ND*	
RBF(50,0)	5.64	67.17 ± 0.33	435.28	5.64	71.03 ± 0.61	450.72	5.65	**78.19** ± 0.27	488.82
RBF(50,0.001)	0.14	28.91 ± 0.26	358.09	0.14	33.72 ± 0.09	381.40	0.14	**40.66** ± 0.09	422.18
RBF(10,0.001)	4.88	63.12 ± 0.32	422.03	4.88	64.71 ± 0.48	435.43	4.88	**72.06** ± 0.25	476.67
RBF(50,0.0001)	0.21	34.51 ± 0.89	378.15	0.21	44.32 ± 0.56	402.54	0.21	**59.13** ± 0.29	445.12
RBF(10,0.0001)	2.24	58.80 ± 1.21	407.90	2.30	63.81 ± 0.42	427.85	2.30	**73.17** ± 0.26	477.25
RT	124.33	14.29 ± 0.45	518.19	125.19	32.44 ± 1.51	534.07	125.19	**65.67** ± 0.53	549.01
LED(50000)	6.29	9.99 ± 0.03	3035.01	6.29	9.99 ± 0.02	3164.40	6.29	**9.99** ± 0.03	3057.94
Waveform(50)	56.64	42.78 ± 10.02	1557.63	60.96	41.27 ± 0.26	1549.53	56.64	**68.05** ± 0.31	1601.11
HYP(10,0.001)	2.19	71.06 ± 4.14	203.96	2.18	78.63 ± 4.36	211.31	2.19	**87.09** ± 1.07	226.83
HYP(10,0.0001)	5.32	73.58 ± 5.25	234.52	5.23	79.61 ± 3.58	247.48	5.13	**87.96** ± 0.82	257.55
	51.88 Acc. 4.52 RAM-Hours			56.70 Acc. 0.46 RAM-Hours			**66.23** Acc. 1.03 RAM-Hours		

Table 3: Comparing accuracy when using only the 10, 20 or 40 best classifiers with trees of 2 attributes. Accuracy is measured as the final percentage of examples correctly classified over the full test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface.

	$k = 3$ (averaging)			k=m-2			k=m-3		
	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.	Time
CovType	240.83	36.67	231276.47	86.22	**82.51**	74636.64		*ND*	
Electricity	0.27	44.54	13.92	0.36	88.86	10.5	0.43	**89.68**	18.08
Poker-Hand	0.72	50.15	614.82	0.3	77.89	391.55	0.71	**78.86**	977.38
CovPokElec		*ND*		219.18	**78.41**	353115.5		*ND*	
RBF(50,0)	21.91	82.03 ± 0.09	1878.91	19.87	88.64 ± 0.13	1247.03	47.56	**89.08** ± 0.12	3180.21
RBF(50,0.001)	0.38	26.27 ± 0.09	1543.45	0.25	52.82 ± 0.06	1254.7	0.59	**53.88** ± 0.06	3059.02
RBF(10,0.001)	18.21	75.33 ± 0.15	1837.22	11.24	82.36 ± 0.12	1263.75	28.06	**83.20** ± 0.06	3246.78
RBF(50,0.0001)	0.68	27.41 ± 0.1	1634.87	0.66	69.62 ± 0.39	1262.16	1.46	**74.47** ± 0.4	3192.08
RBF(10,0.0001)	5.14	63.72 ± 5.78	1784.83	2.21	85.72 ± 0.11	1315.12	4.88	**86.91** ± 0.08	3277.38
RT	260.14	51.7 ± 0.34	1480.11	39.75	92.62 ± 0.2	825.85	79.61	**93.35** ± 0.21	2040.98
LED(50000)	68.95	72.83 ± 0.05	45986.89	5.5	**73.13** ± 0.06	20162.39	45.12	73.06 ± 0.1	146666.31
Waveform(50)	181.73	83.01 ± 0.31	18996.19	129.46	**85.51** ± 0.29	8145.45	411.46	85.44 ± 0.26	61121.98
HYP(10,0.001)	7.69	53.4 ± 1.56	789.37	5.99	90.91 ± 0.26	577.59	13.27	**91.26** ± 0.24	1455.89
HYP(10,0.0001)	16.19	78.26 ± 2.96	843.25	13.55	91.63 ± 0.1	628.6	31.99	**91.88** ± 0.08	1541.61
	57.33 Acc. 68.9 RAM-Hours			81.47 Acc. 67.4 RAM-Hours			**82.59** Acc. 41.46 RAM-Hours		

Table 4: Performance for $k = 3$ when probability estimates are simply averaged. Also shown: stacking using trees with m-2 and m-3 attributes each. Accuracy is measured as the final percentage of examples correctly classified over the full test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface.

6. Conclusions

This paper has presented a new method for data stream classification that is based on combining an exhaustive ensemble of limited-attribute-set tree classifiers using stacking. The approach generates tree classifiers for all possible attribute subsets of a given user-specified size k. For $k = 2$, this incurs an extra factor of k in time complexity compared to building a single tree grown from all attributes. For larger values of k, the method is only practical for datasets with a small number of attributes (unless k is chosen to be close to the number of attributes). However, even if limited to $k = 2$, the method appears to be a promising candidate for high-accuracy data stream classification on practical data streams. Our results on real-world datasets show excellent classification accuracy.

Part of our method is an approach for resetting the learning rate for the perceptron meta-classifiers that are used to combine the trees' predictions. This method, based on the `ADWIN` change detector, is employed because the meta-data stream is not identically distributed: the trees' predictions improve over time. When change is detected, the learning rate is reset to its initial, larger value, so that meta learning quickly adapts to the improved predictions of the base models.

The `ADWIN` change detector is also used to reset ensemble members when their predictive accuracy degrades significantly, in which case the learning rate is reset also. This makes it possible to use the ensemble classifier for evolving data streams where the distribution changes over time. Our empirical results based on evolving data show that the proposed method can indeed cope successfully with change.

We have also empirically verified that it is important to use stacking to learn how to combine the tree classifiers' predictions in an appropriate manner: simple averaging yields very poor accuracy. This is not surprising because not all attribute subsets are equally relevant. Our initial experiments with ensemble pruning, discarding the least important ensemble members, did not prove successful. In future work, we would like to investigate smarter pruning techniques and methods for pre-selecting attribute subsets, e.g. based on maximizing pairwise Hamming distance between indicator vectors.

References

Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *SDM*, 2007.

Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, and Ricard Gavaldà. Improving adaptive bagging methods for evolving data streams. In *ACML*, pages 23–37, 2009a.

Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *KDD*, pages 139–148, 2009b.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive Online Analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010a.

Albert Bifet, Geoffrey Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *ECML/PKDD*, pages 135–150, 2010b.

Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. In *PAKDD*, pages 299–310, 2010c.

Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *KDD*, pages 71–80, 2000.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

João Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *KDD*, pages 523–528, 2003.

João Gama, Pedro Medas, Gladys Castillo, and Pedro Pereira Rodrigues. Learning with drift detection. In *SBIA*, pages 286–295, 2004.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *KDD*, pages 329–338, 2009.

Michael Harries. Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales, 1999.

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. Stress-testing Hoeffding trees. In *PKDD*, pages 495–502, 2005.

Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD*, pages 97–106, 2001.

Richard Kirkby. *Improving Hoeffding Trees*. PhD thesis, University of Waikato, November 2007.

Nikunj C. Oza and Stuart J. Russell. Experimental comparisons of online and batch versions of bagging and boosting. In *KDD*, pages 359–364, 2001a.

Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. In *AISTATS*, pages 105–112, 2001b.

David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.