Leveraging Bagging for Evolving Data Streams

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer

University of Waikato, Hamilton, New Zealand
{abifet,geoff,bernhard}@cs.waikato.ac.nz

Abstract. Bagging, boosting and Random Forests are classical ensemble methods used to improve the performance of single classifiers. They obtain superior performance by increasing the accuracy and diversity of the single classifiers. Attempts have been made to reproduce these methods in the more challenging context of evolving data streams. In this paper, we propose a new variant of bagging, called *leveraging bagging*. This method combines the simplicity of bagging with adding more randomization to the input, and output of the classifiers. We test our method by performing an evaluation study on synthetic and real-world datasets comprising up to ten million examples.

1 Introduction

Data Stream real time analytics are needed to manage data generated at an increasing rate from sensor applications, measurements in network monitoring and traffic management, log records or click-streams in web exploring, manufacturing processes, call detail records, email, blogging, twitter posts and others. In fact, all data generated can be considered as streaming data or as a snapshot of streaming data, since it is obtained from an interval of time.

In the data stream model, data arrive at high speed, and algorithms that process them must do so under very strict constraints of space and time. Consequently, data streams pose several challenges for data mining algorithm design. First, algorithms must make use of limited resources (time and memory). Second, they must deal with data whose nature or distribution changes over time.

Bagging and Boosting are ensemble methods used to improve the accuracy of classifier methods. Non-streaming bagging [7] builds a set of M base models, training each model with a bootstrap sample of size N created by drawing random samples with replacement from the original training set. Each base model's training set contains each of the original training example K times where $P(K = k)$ follows a binomial distribution. This binomial distribution for large values of N tends to a Poisson(1) distribution, where Poisson(1)$= \exp(-1)/k!$. Using this fact, Oza and Russell [25,24] proposed *Online Bagging*, an online method that instead of sampling with replacement, gives each example a weight according to Poisson(1).

Boosting algorithms combine multiple base models to obtain a small generalization error. Non-streaming boosting builds a set of models sequentially, with

J.L. Balcázar et al. (Eds.): ECML PKDD 2010, Part I, LNAI 6321, pp. 135–150, 2010.

the construction of each new model depending on the performance of the previously constructed models. The intuitive idea of boosting is to give more weight to misclassified examples, and reducing the weight of the correctly classified ones.

From studies appearing in the literature [25,24,6], Online Bagging seems to perform better than online boosting methods. Why bagging outperforms boosting in the data stream setting is still an open question. Adding more random weight to all instances seems to improve accuracy more than adding weight to misclassified instances. In this paper we focus on randomization as a powerful tool to increase accuracy and diversity when constructing an ensemble of classifiers. There are three ways of using randomization:

- Manipulating the input data
- Manipulating the classifier algorithms
- Manipulating the output targets

In this paper we focus on randomizing the input data and the output prediction of online bagging. The paper is structured as follows: related work is presented in Section 2. Leveraging bagging is discussed in Section 3. An experimental evaluation is conducted in Section 4. Finally, conclusions and suggested items for future work are presented in Section 5.

2 Related Work

Breiman [7] introduced bagging classification using the notion of an *order-correct* learner. An order-correct learner ϕ at the input x is a predictor that if input x results in a class more often than any other class, then ϕ will also predict this class at x more often than any other class. An order-correct learner is not necessarily an accurate predictor but its aggregated predictor is optimal. If a predictor is good because it is order-correct for most inputs x then aggregation can transform it into a nearly optimal predictor. The vital element to gain accuracy is the instability of the prediction method. A learner is unstable if a small change in the input data leads to large changes in the output.

Friedman [16] explained that bagging works by reducing variance without changing the bias. There are several definitions of bias and variance for classification, but the common idea is that bias measures average error over many different training sets, and variance measures the additional error due to the variation in the model produced by using different training sets.

Domingos [13] claimed that Breiman's line of reasoning is limited, since we may never know *a priori* whether a learner is order-correct for a given example or not, and what regions of the instance space will be order-correct or not. He explained bagging's success showing that bagging works by effectively changing a single-model learner to another single-model learner, with a different implicit prior distribution over models, one that is less biased in favor of simple models.

Some work in the literature shows that bagging asymptotically performs some smoothing on the estimate. Friedman and Hall [15] used an asymptotic truncated Taylor series of the estimate to show that in the limit of infinite samples, bagging reduces the variance of non-linear components.

Bühlmann and Yu [10], analyzed bagging using also asymptotic limiting distributions, and they proposed *subagging* as a less expensive alternative to bagging. Subagging uses subsampling as an alternative aggregation scheme. They claimed that subagging is as accurate as bagging but uses less computation.

Grandvalet[19] explained the performance of bagging by the goodness and badness of highly influential examples, in situations where the usual variance reduction argument is questionable. He presented an experiment showing that bagging increases the variance of decision trees, and claimed that bagging does not simply reduce variance in its averaging process.

In [6] two new state-of-the-art bagging methods were presented: ASHT Bagging using trees of different sizes, and `ADWIN` Bagging using a change detector to decide when to discard underperforming ensemble members.

Breiman [8] proposed Random Forests as a method to use randomization on the input and on the internal construction of the decision trees. Random Forests are ensembles of trees with the following characteristics: the input training set is obtained by sampling with replacement, the nodes of the tree only may use a fixed number of random attributes to split, and the trees are grown without pruning. Abdulsalam et al. [1] presented a streaming version of random forests and Saffari et al. [26] presented an online version.

3 Leveraging Bagging

In this section, we present a new online leveraging bagging algorithm, improving Online Bagging of Oza and Russell. The pseudo-code of Online Bagging of Oza and Russell is listed in Algorithm 1.

We leverage the performance of bagging, with two randomization improvements: increasing resampling and using output detection codes.

Resampling with replacement is done in Online Bagging using Poisson(1). There are other sampling mechanisms:

- Lee and Clyde [23] uses the Gamma distribution (Gamma(1,1)) to obtain a Bayesian version of Bagging. Note that Gamma(1,1) is equal to Exp(1).
- Bulhman and Yu [10] proposes subagging, using resampling without replacement.

Algorithm 1. Oza and Russell's *Online Bagging* for M models

1: Initialize base models h_m for all $m \in \{1, 2, ..., M\}$
2: **for all** training examples **do**
3: **for** $m = 1, 2, ..., M$ **do**
4: Set $w = Poisson(1)$
5: Update h_m with the current example with weight w
6: **anytime output:**
7: **return** hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$

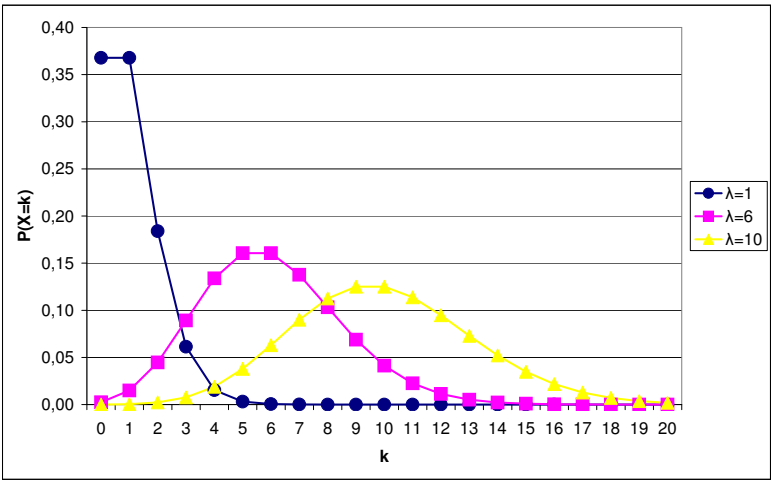


Fig. 1. Poisson Distribution

Our proposal is to increase the weights of this resampling using a larger value λ to compute the value of the Poisson distribution. The Poisson distribution is used to model the number of events occurring within a given time interval.

Figure 1 shows the probability function mass of the distribution of Poisson for several values of λ. The mean and variance of a Poisson distribution is λ. For $\lambda = 1$ we see that 37% of the values are zero, 37% are one, and 26% are values greater than one. Using a weight of Poisson(1) we are taking out 37% of the examples, and repeating 26% of the examples, in a similar way to non streaming bagging. For $\lambda = 6$ we see that 0.25% of the values are zero, 45% are lower than six, 16% are six, and 39% are values greater than six. Using a value of $\lambda > 1$ for Poisson(λ) we are increasing the diversity of the weights and modifying the input space of the classifiers inside the ensemble. However, the optimal value of λ may be different for each dataset.

Our second improvement is to add randomization at the output of the ensemble using output codes. Dietterich and Bakiri [12] introduced a method based on error-correcting output codes, which handles multiclass problems using only a binary classifier. The classes assigned to each example are modified to create a new binary classification of the data induced by a mapping from the set of classes to {0,1}. A variation of this method by Schapire [27] presented a form of boosting using output codes.

We assign to each class a binary string of length n and then build an ensemble of n binary classifiers. Each of the classifiers learns one bit for each position in this binary string. When a new instance arrives, we assign x to the class whose binary code is closest. We can view an error-correcting code as a form of voting in which a number of incorrect votes can be corrected.

We use random output codes instead of deterministic codes. In standard ensemble methods, all classifiers try to predict the same function. However, using

output codes each classifier will predict a different function. This may reduce the effects of correlations between the classifiers, and increase diversity of the ensemble.

We implement random output codes in the following way: we choose for each classifier m and class c a binary value $\mu_m(c)$ in a uniform, independent, and random way. We ensure that exactly half of the classes are mapped to 0. The output of the classifier for an example is the class which has more votes of its binary mapping classes. Table 1 shows an example for an ensemble of 6 classifiers in a classification task of 3 classes.

Table 1. Example matrix of random output codes for 3 classes and 6 classifiers

	Class 1	Class 2	Class 3
Classifier 1	0	0	1
Classifier 2	0	1	1
Classifier 3	1	0	0
Classifier 4	1	1	0
Classifier 5	1	0	1
Classifier 6	0	1	0

We use the strategy of [6] to deal with concept drift. `ADWIN` [3] is a change detector and estimator that solves in a well-specified way the problem of tracking the average of a stream of bits or real-valued numbers. `ADWIN` keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window".

`ADWIN` is parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, inherent to all algorithms dealing with random processes.

Also important for our purposes, `ADWIN` does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique.

Algorithm 2. *Leveraging Bagging* for M models

1: Initialize base models h_m for all $m \in \{1, 2, ..., M\}$
2: Compute coloring $\mu_m(y)$
3: **for all** training examples (x, y) **do**
4: **for** $m = 1, 2, ..., M$ **do**
5: Set $w = Poisson(\lambda)$
6: Update h_m with the current example with weight w and class $\mu_m(y)$
7: **if** `ADWIN` detects change in error of one of the classifiers **then**
8: Replace classifier with higher error with a new one
9: **anytime output:**
10: **return** hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = \mu_t(y))$

This means that it keeps a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item.

Algorithm 2 shows the pseudo-code of our Leveraging Bagging. First we build a matrix with the values of μ for each classifier and class. For each new instance that arrives, we give it a random weight of $Poisson(k)$. We train the classifier with this weight, and when a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble. To predict the class of an example, we compute for each class c the sum of the votes for $\mu(c)$ of all the ensemble classifiers, and we output as a prediction the class with the most votes.

4 Comparative Experimental Evaluation

Massive **O**nline **A**nalysis (MOA) [4] is a software environment for implementing algorithms and running experiments for online learning from data streams. All algorithms evaluated in this paper were implemented in the Java programming language by extending the MOA software.

We use the experimental framework for concept drift presented in [6]. Considering data streams as data generated from pure distributions, we can model a concept drift event as a weighted combination of two pure distributions that characterizes the target concepts before and after the drift. This framework defines the probability that a new instance of the stream belongs to the new concept after the drift based on the sigmoid function.

Definition 1. *Given two data streams a, b, we define $c = a \oplus^{W}_{t_0} b$ 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* $\Pr[c(t) = a(t)] = 1 - \Pr[c(t) = b(t)]$.

In order to create a data stream with multiple concept changes, we can build new data streams joining different concept drifts, i. e. $(((a \oplus^{W_0}_{t_0} b) \oplus^{W_1}_{t_1} c) \oplus^{W_2}_{t_2} d) \ldots$.

4.1 Datasets for Concept Drift

Synthetic data has several advantages – it is easier to reproduce and there is little cost in terms of storage and transmission. For this paper we use the data generators most commonly found in the literature.

SEA Concepts Generator. This artificial dataset contains abrupt concept drift, first introduced in [28]. It is generated using three attributes, where only the two first attributes are relevant. All the attributes have values between 0 and 10. The points of the dataset are divided into 4 blocks with different concepts. In each block, the classification is done using $f_1 + f_2 \leq \theta$, where f_1 and f_2 represent the first two attributes and θ is a threshold value. The most frequent values are 9, 8, 7 and 9.5 for the data blocks. In our framework, SEA concepts are defined as follows:

$$(((SEA_9 \oplus^W_{t_0} SEA_8) \oplus^W_{2t_0} SEA_7) \oplus^W_{3t_0} SEA_{9.5})$$

Rotating Hyperplane. This data was used as a testbed for CVFDT versus VFDT in [22]. Examples for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labeled positive, and examples for which $\sum_{i=1}^{d} w_i x_i < w_0$ are labeled negative. Hyperplanes are useful for simulating time-changing concepts, because we can change the orientation and position of the hyperplane in a smooth manner by changing the relative size of the weights.

Random RBF Generator. This generator was devised to offer an alternate complex concept type that is not straightforward to approximate with a decision tree model. The RBF (Radial Basis Function) generator works as follows: A fixed number of random centroids are generated. Each center has a random position, a single standard deviation, class label and weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weight are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. Drift is introduced by moving the centroids with constant speed.

LED Generator. This data source originates from the CART book [9]. An implementation in C was donated to the UCI [2] machine learning repository by David Aha. The goal is to predict the digit displayed on a seven-segment LED display, where each attribute has a 10% chance of being inverted. The particular configuration of the generator used for the experiments (led) produces 24 binary attributes, 17 of which are irrelevant.

4.2 Real-World Data

The UCI machine learning repository [2] contains some real-world benchmark data for evaluating machine learning techniques. We consider three of the largest: Forest Covertype, Poker-Hand, and Electricity.

Forest Covertype. Contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains $581,012$ instances and 54 attributes, and it has been used in several papers on data stream classification [18,25].

Poker-Hand. Consists of $1,000,000$ instances and 11 attributes. Each record of the Poker-Hand dataset is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one class attribute that describes the "Poker Hand".

Electricity is another widely used dataset described by M. Harries [20] and analysed by Gama [17]. This data was collected from the Australian New South Wales Electricity Market. In this market, prices are not fixed and are affected by demand and supply of the market. They are set every five minutes. The ELEC dataset contains $45,312$ instances. The class label identifies the change of the price relative to a moving average of the last 24 hours.

Table 2. Comparison of Hoeffding Tree, Online bagging and ADWIN bagging. Accuracy is measured as the final percentage of examples correctly classified over the 1 or 10 million test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface.

	Hoeffding Tree			Online Bagging			ADWIN Bagging		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
RBF(0,0)	0.97	88.10 ± 0.34	141.37	27.35	**91.59** ± 0.11	2656.28	27.16	91.58 ± 0.11	3311.22
RBF(50,0.001)	0.97	30.93 ± 0.03	178.30	25.48	32.89 ± 0.04	2894.04	0.25	**41.64** ± 0.04	5481.12
RBF(10,0.001)	0.97	80.23 ± 0.15	137.30	26.90	83.39 ± 0.10	2759.74	26.07	**83.41** ± 0.08	3579.72
RBF(50,0.0001)	0.98	35.25 ± 0.09	166.22	27.85	44.48 ± 0.07	3245.18	0.73	**60.54** ± 0.07	5519.06
RBF(10,0.0001)	0.97	80.95 ± 0.14	132.80	27.86	84.59 ± 0.12	2682.15	26.83	**84.78** ± 0.11	3481.96
LED(50000)	1.94	68.50 ± 0.29	22.99	50.93	69.00 ± 0.16	544.15	5.10	**73.08** ± 0.08	541.42
SEA(50)	0.49	86.48 ± 0.06	5.32	12.13	86.83 ± 0.06	86.66	10.23	**87.59** ± 0.29	107.13
SEA(50000)	0.49	86.45 ± 0.07	5.55	12.12	86.79 ± 0.06	91.43	6.32	**88.32** ± 0.14	99.49
HYP(10,0.001)	1.45	80.70 ± 1.44	85.62	57.85	83.05 ± 1.49	2017.98	28.08	**90.74** ± 0.21	2822.05
HYP(10,0.0001)	1.26	84.12 ± 0.75	85.43	48.91	85.88 ± 0.80	1913.85	36.05	**91.23** ± 0.12	3145.88
CovType	2.65	80.70	27.46	59.83	83.93	345.62	4.80	**84.91**	486.00
Electricity	0.09	79.20	0.98	3.12	82.66	5.88	1.16	**84.51**	7.13
Poker	0.59	77.10	11.62	13.85	**82.29**	171.13	0.39	70.68	202.99
CovPokElec	5.69	77.65	62.63	138.09	**82.67**	1247.47	7.74	76.40	1367.09
	74.03 Acc.			77.15 Acc.			**79.24** Acc.		
	0.01 RAM-Hours			2.98 RAM-Hours			1.48 RAM-Hours		
	2.86 avg. rank			1.79 avg. rank			1.36 avg. rank		

Nemenyi significance: Online Bagging$\succ$ Hoeffding Tree; ADWIN Bagging $\succ$ Hoeffding Tree;

Table 3. Comparison of ADWIN Half subagging, ADWIN subagging, and ADWIN bagging using all instances. Accuracy is measured as the final percentage of examples correctly classified over the 1 or 10 million test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface.

	ADWIN Half subagging			ADWIN Subagging			ADWIN Bagging WT		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
RBF(0,0)	23.99	91.22 ± 0.09	3986.69	24.07	**91.43** ± 0.11	3896.28	31.50	91.36 ± 0.16	3198.21
RBF(50,0.001)	0.16	43.81 ± 0.03	4887.80	0.16	**43.93** ± 0.02	5244.13	0.45	43.66 ± 0.03	6627.15
RBF(10,0.001)	23.28	83.14 ± 0.09	4133.80	23.26	**83.29** ± 0.08	4132.29	30.11	83.04 ± 0.16	3353.40
RBF(50,0.0001)	0.37	56.61 ± 0.05	4927.20	0.41	57.07 ± 0.06	5296.05	2.55	**72.31** ± 0.08	6682.20
RBF(10,0.0001)	23.14	85.07 ± 0.16	4090.61	23.18	**85.12** ± 0.12	4091.19	32.88	84.57 ± 0.11	3272.87
LED(50000)	1.50	73.05 ± 0.07	478.39	1.83	**73.11** ± 0.08	515.13	12.28	73.06 ± 0.07	616.17
SEA(50)	4.83	87.43 ± 0.27	82.08	6.13	87.51 ± 0.28	98.79	19.95	**87.88** ± 0.36	153.28
SEA(50000)	2.91	87.98 ± 0.17	75.05	3.60	88.17 ± 0.18	92.18	12.46	**88.55** ± 0.24	136.75
HYP(10,0.001)	17.81	90.27 ± 0.17	2510.85	20.50	90.35 ± 0.17	2739.65	50.60	**90.59** ± 0.20	3786.17
HYP(10,0.0001)	22.51	90.65 ± 0.09	2605.25	23.49	90.73 ± 0.11	2886.55	59.75	**91.10** ± 0.12	4075.18
CovType	1.82	82.03	507.03	2.18	82.55	525.32	17.74	**88.47**	505.22
Electricity	0.57	82.45	6.91	0.62	83.38	7.42	1.99	**86.67**	8.82
Poker	0.38	69.24	215.07	0.38	69.99	230.15	1.61	**77.35**	213.41
CovPokElec	3.27	74.16	1647.06	3.37	74.94	1411.78	22.30	**82.18**	1527.23
	78.36 Acc.			78.68 Acc.			**81.49** Acc.		
	1.04 RAM-Hours			1.13 RAM-Hours			2.74 RAM-Hours		
	2.79 avg. rank			1.64 avg. rank			1.57 avg. rank		

Nemenyi significance: ADWIN Subagging$\succ$ADWIN Half subagging; ADWIN Bagging WT$\succ$ADWIN Half subagging;

We use normalized versions of these datasets, so that the numerical values are between 0 and 1. With the Poker-Hand dataset, the cards are not ordered, i.e. a hand can be represented by any permutation, which makes it very hard for propositional learners, especially for linear ones. We use a modified version, where cards are sorted by rank and suit, and have removed duplicates. This dataset loses about 171, 799 examples, and comes down to 829, 201 examples.

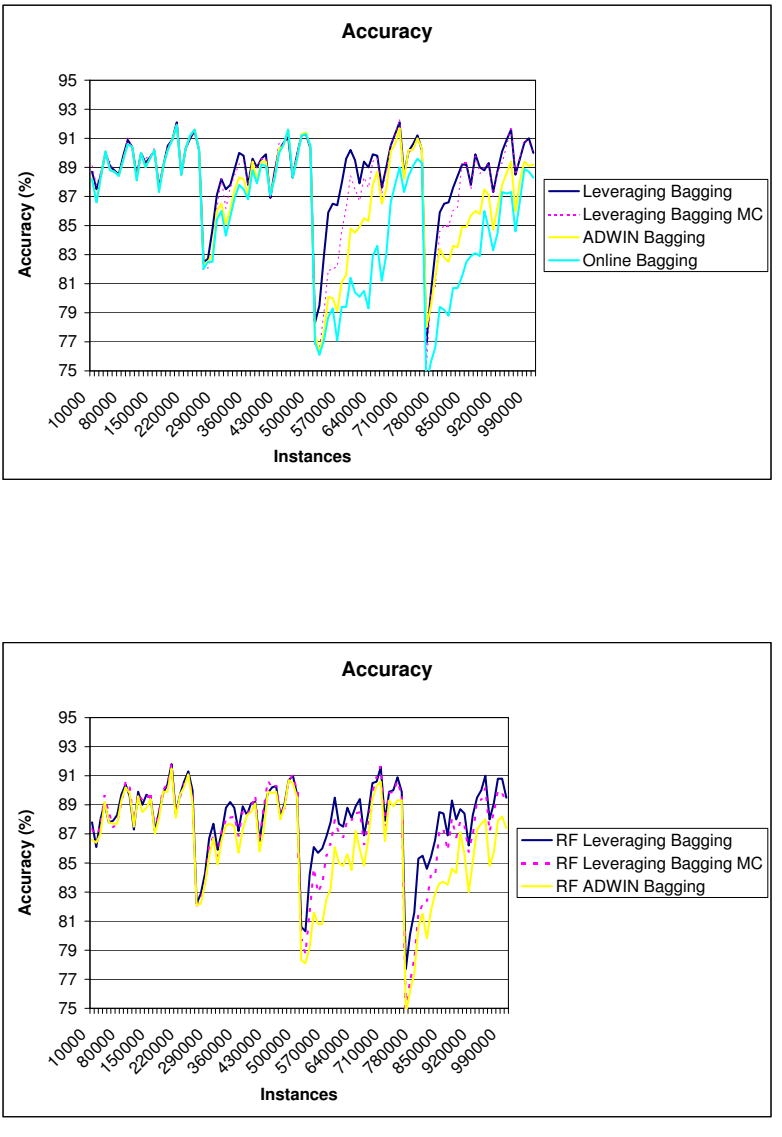


Fig. 2. Accuracy on the SEA data with three concept drifts

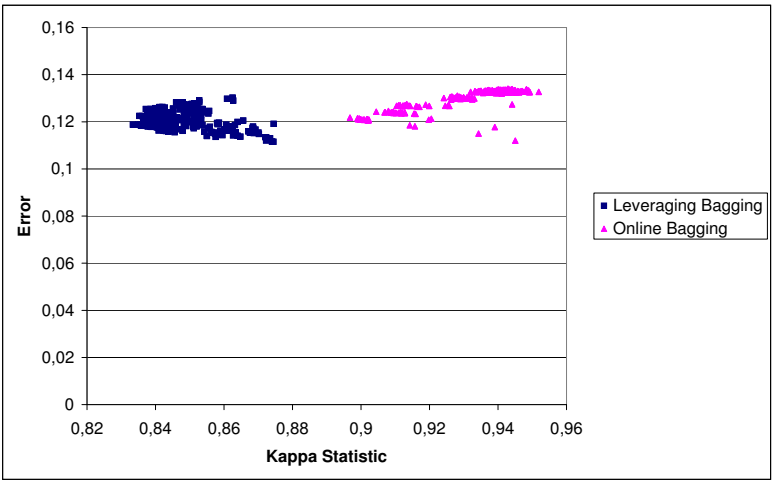


Fig. 3. Kappa-Error diagrams for Leveraging Bagging and Online bagging (bottom) on on the SEA data with three concept drifts, plotting 576 pairs of classifiers

These datasets are small compared to synthetic datasets we consider. Another important fact is that we do not know when drift occurs or indeed if there is any drift. We may simulate concept drift, joining the three datasets, merging attributes, and supposing that each dataset corresponds to a different concept

$$\text{CovPokElec} = (\text{CoverType} \oplus^{5,000}_{581,012} \text{Poker}) \oplus^{5,000}_{1,000,000} \text{ELEC}$$

As all examples need to have the same number of attributes, we simply concatenate all the attributes, and set the number of classes to the maximum number of classes of all the datasets.

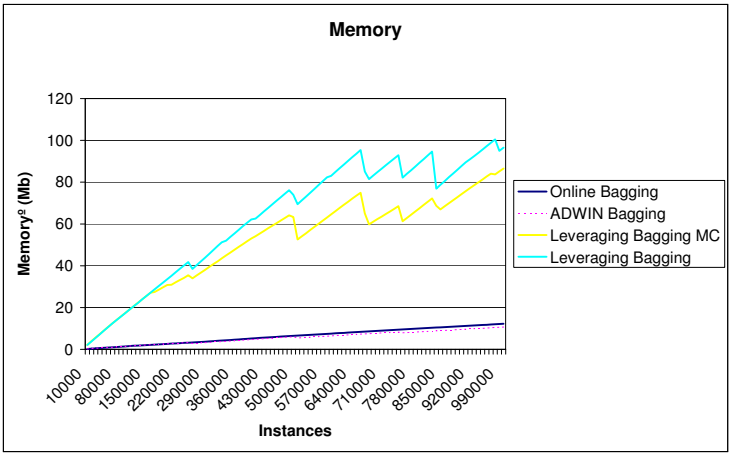
4.3 Results

We ran three experimental evaluations to test our new leveraging method using 25 classifiers. In the first, in order to understand why online bagging works, we compare the original Online Bagging, with the `ADWIN` Bagging presented in [6], and with two different resampling stategies

- half subagging
- without replacement or subagging
- without taking out all instances (WT)

The second experiment measures accuracy improvement of leveraging bagging. And finally, the third experiment compares our method against online Random Forests.

We use the datasets explained in the previous sections for evaluation. The experiments were performed on 2.66 GHz Core 2 Duo E6750 machines with 4



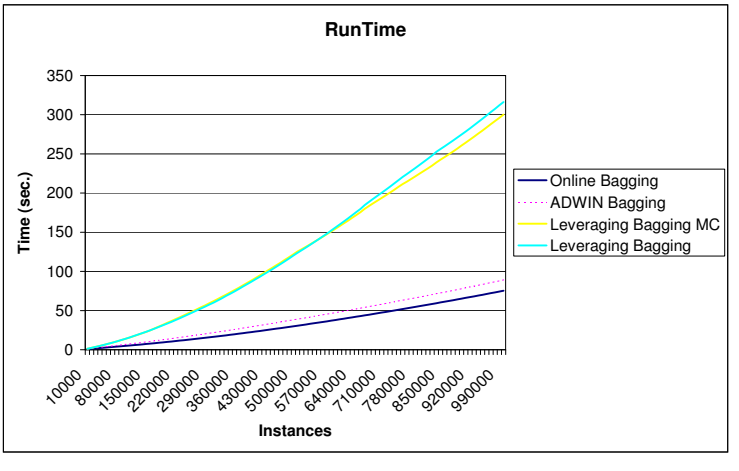


Fig. 4. Runtime and memory on the SEA data with three concept drifts

GB of memory. The evaluation methodology used was Interleaved Test-Then-Train on 10 runs and 25 classifiers: every example was used for testing the model before using it to train. This interleaved test followed by train procedure was carried out on 10 million examples from the hyperplane and RandomRBF datasets, and one million examples from the SEA dataset. The parameters of these streams are the following:

- RBF(x,v): RandomRBF data stream of 5 classes with x centroids moving at speed v.
- HYP(x,v): Hyperplane data stream of 5 classes with x attributes changing at speed v.
- SEA(v): SEA dataset, with length of change v.
- LED(v): LED dataset, with length of change v.

The Nemenyi test [11] is used for computing significance: it is an appropriate test for comparing multiple algorithms over multiple datasets, being based on the average ranks of the algorithms across all datasets. We use a p-value of 0.05. Under the Nemenyi test, $\{x,y\} \succ \{z\}$ indicates that algorithms x and y are statistically significantly more likely to be more favourable than z.

In [5] we introduced the use of RAM-Hours as an evaluation measure of the resources used by streaming algorithms. Every GB of RAM deployed for 1 hour equals one RAM-Hour.

Tables 2, 3 and 4 report the final accuracy, and speed of the classification models induced on the synthetic data and the real datasets: Forest CoverType, Poker Hand, Electricity and CovPokElec. 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. All experiments are performed using leveraging bagging with Poisson(6), since empirically this was determined to be the best value. We implemented Online Bayesian Bagging, but we don't report the results as they are similar to those using Poisson(1).

We use as a base learner for our experiments the *Hoeffding Naive Bayes Tree* (**hnbt**). Hoeffding trees [14] are state-of-the-art in classification for data streams and they 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. A Hoeffding Naive Bayes Tree [21] works by performing a naive Bayes prediction per training instance, and 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.

Table 2 reports the accuracy, speed and memory of a Hoeffding Naive Bayes Tree(**hnbt**), compared with online bagging of Hoeffding Naive Bayes Tree and `ADWIN` bagging of Hoeffding Naive Bayes Trees. We observe that online bagging improves the accuracy of a single Hoeffding Naive Bayes Tree from 74.03% to 77.15%. `ADWIN` bagging improves this result getting 79.24%. In terms of memory and speed, the single Hoeffding Naive Bayes Tree is much faster and needs less memory.

Table 4. Comparison of Leveraging Bagging without using Random Output Codes, Leveraging Bagging using Random Output Codes, and Leveraging Bagging giving more weight to missclassified examples. The best individual accuracies are indicated in boldface.

	Leveraging Bagging			Leveraging Bagging MC			Leveraging Bagging ME		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
RBF(0,0)	56.51	91.05 ± 0.06	2862.03	133.63	**91.07** ± 0.03	3980.25	23.22	90.11 ± 0.38	5492.83
RBF(50,0.001)	3.31	**58.21** ± 0.05	7333.89	181.04	56.64 ± 0.07	6511.17	0.17	44.67 ± 0.02	7674.30
RBF(10,0.001)	61.77	82.45 ± 0.07	2997.72	212.74	82.62 ± 0.06	6518.94	22.89	**83.77** ± 0.15	5153.45
RBF(50,0.0001)	21.55	**80.48** ± 0.02	7421.81	73.84	78.42 ± 0.04	5155.42	0.38	58.58 ± 0.07	7375.62
RBF(10,0.0001)	64.86	83.94 ± 0.07	2899.91	249.36	86.00 ± 0.17	7847.99	22.30	**86.35** ± 0.24	5180.18
LED(50000)	27.64	73.10 ± 0.09	714.17	121.15	71.67 ± 0.16	491.14	1.56	**73.11** ± 0.08	647.63
SEA(50)	92.88	**88.65** ± 0.15	354.51	96.59	88.50 ± 0.18	362.56	1.27	87.64 ± 0.16	58.92
SEA(50000)	75.12	**88.69** ± 0.11	324.51	80.78	88.51 ± 0.10	336.27	1.02	87.31 ± 0.13	59.10
HYP(10,0.001)	409.89	88.66 ± 0.38	11307.98	189.75	88.01 ± 0.43	5537.68	3.89	**91.02** ± 0.07	2047.55
HYP(10,0.0001)	405.58	89.36 ± 0.13	11838.65	207.66	88.63 ± 0.27	5873.24	4.28	**91.19** ± 0.05	2014.43
CovType	49.45	91.29	559.57	43.88	**92.53**	368.29	1.62	90.96	479.49
Electricity	6.23	**88.41**	11.11	6.85	87.98	11.56	0.16	**88.41**	5.88
Poker	32.42	98.03	194.48	47.25	**98.76**	194.62	0.27	75.87	208.08
CovPokElec	167.47	95.23	1610.18	185.44	**95.83**	1204.34	2.25	81.80	1360.60
	85.54 Acc. 20.17 RAM-Hours 1.79 avg. rank			85.37 Acc. 22.04 RAM-Hours 2.00 avg. rank			80.77 Acc. 0.87 RAM-Hours 2.14 avg. rank		

We ran experiments to test three different bagging strategies: subagging (resampling without replacement), half subagging (resampling without replacement half of the instances), and bagging without taking out any instance. We implement this third strategy using $1+Poisson(1)$ instead of Poisson. We tested the three strategies on `ADWIN` bagging. Table 3 shows, for these bagging strategies, their accuracy, speed and memory. We observe that using subagging we get faster methods but less accurate. If we use all instances, it seems that we improve accuracy but not speed or the memory used.

The learning curves and model growth curves for the Sea dataset are plotted in Figures 2 and 4. We observe that for this data stream the new leveraging bagging methods need more time and memory than the other methods, but they are more accurate.

We use the Kappa statistic κ [6] to show how using Leveraging Bagging with $\lambda = 6$, we increase the diversity of the ensemble. When two classifiers agree on every example then $\kappa = 1$, and when their predictions coincide purely by chance, then $\kappa = 0$.

The Kappa-Error diagram is a scatterplot where each point corresponds to a pair of classifiers. The x coordinate of the pair is the κ value for the two classifiers. The y coordinate is the average of the error rates of the two classifiers.

Figure 3 shows the Kappa-Error diagram for the SEA dataset with three concept drifts and 25 classifiers. We observe that for this dataset the Kappa statistic for Leveraging Bagging is lower than for Online Bagging, showing the higher diversity of the output of the classifiers of the Leveraging Bagging method.

Table 4 reports the accuracy, speed and memory of the new Levering Bagging methods using **hnbt**. We compare Levering Bagging with Levering Bagging MC without using Random Ouput Codes, and Levering Bagging ME giving more weight to missclassified examples. In this last method, if an instance is

Table 5. Comparison of methods using Random Forests: Leveraging Bagging without using Random Output Codes, Online Bagging, and ADWIN bagging. Accuracy is measured as the final percentage of examples correctly classified over the 1 or 10 million test/train interleaved evaluation. Time is measured in seconds, and memory in MB. The best individual accuracies are indicated in boldface.

	RF Leveraging Bagging			RF Online Bagging			RF ADWIN Bagging		
	Time	Acc.	Mem.	Time	Acc.	Mem.	Time	Acc.	Mem.
RBF(0,0)	45.75	**90.70** ± 0.05	2193.41	24.99	90.33 ± 0.11	1624.37	24.59	90.31 ± 0.10	1939.25
RBF(50,0.001)	3.54	**55.56** ± 0.06	1835.02	24.32	31.31 ± 0.03	1453.76	0.11	34.18 ± 0.02	1177.31
RBF(10,0.001)	49.21	**82.13** ± 0.06	2152.77	24.84	81.84 ± 0.09	1691.17	23.93	81.81 ± 0.08	2020.98
RBF(50,0.0001)	23.92	**77.77** ± 0.03	2478.19	24.64	39.72 ± 0.08	1783.48	0.39	48.53 ± 0.11	1497.54
RBF(10,0.0001)	51.93	**83.45** ± 0.07	2181.24	24.92	82.74 ± 0.05	1702.03	23.95	82.73 ± 0.05	2065.70
LED(50000)	20.90	**67.83** ± 0.74	286.48	11.46	60.22 ± 0.71	148.12	3.02	66.12 ± 0.66	166.40
SEA(50)	189.36	**87.86** ± 0.13	760.45	63.31	86.86 ± 0.06	176.44	60.39	86.98 ± 0.06	190.24
SEA(50000)	186.40	**87.74** ± 0.09	728.12	63.30	86.78 ± 0.06	168.49	59.96	86.88 ± 0.06	185.48
HYP(10,0.001)	143.84	**86.09** ± 0.36	5059.25	27.86	80.45 ± 1.47	1484.37	25.39	83.18 ± 0.68	1562.69
HYP(10,0.0001)	132.58	**86.73** ± 0.37	4826.40	27.73	83.43 ± 0.89	1460.90	26.30	84.08 ± 0.57	1607.33
CovType	6.98	**87.81**	162.88	16.05	74.71	130.50	1.07	76.47	140.29
Electricity	2.55	**86.85**	5.52	1.36	80.08	2.94	0.60	82.44	4.32
Poker	7.38	**75.72**	92.94	22.69	74.07	88.94	0.44	65.96	81.23
CovPokElec	11.12	**73.43**	448.82	32.28	68.22	383.78	2.11	69.73	429.11
	80.69 Acc. 5.51 RAM-Hours 1.00 avg. rank			72.91 Acc. 1.30 RAM-Hours 2.71 avg. rank			74.24 Acc. 0.89 RAM-Hours 2.29 avg. rank		

Nemenyi significance: RF Leveraging Bagging≻RF Online Bagging; RF Leveraging Bagging≻RF ADWIN Bagging;

misclassified it is accepted with a weight of one. If not, it is accepted with probability $e_T/(1-e_T)$, where the error estimate e_T is computed as a smoothed version of the proportion of misclassified examples using the estimation of ADWIN that is monitoring the error. We observe that the accuracy of the two Leveraging bagging methods are similar and that they are 6% more accurate than the ADWIN bagging. When leveraging bagging is used to give more weight to missclassified examples, it does not seem to increase accuracy. However it improves the need for RAM-Hours, so the Leveraging Bagging ME is a very good classifier when resources are scarce.

Finally, we compare our methods with Random Forests. We build Random Forests using Hoeffding Naive Bayes Trees in the following way: let n be the number of attributes, we select for each node, $\sqrt(n)$ attributes randomly, and we only keep statistics of these attributes. The splits of the node are made using the best of these attributes, and the predictions at the leaves are made using only the statistics of these attributes.

We compare using the three bagging methods: online bagging, ADWIN bagging, and leveraging bagging using 25 classifiers. The results are shown in Table 5. We see that RandomForests are, for many datasets, twice as fast and use half of the memory. However their accuracy is 5% below that of using standard Hoeffding Naive Bayes trees. To obtain the same accuracy as the Hoeffding Naive Bayes trees, we only need to increase the number of classifiers of the ensemble. We can observe that using our new leveraging bagging we increase the accuracy of Random Forests.

5 Conclusions

We have presented a new leveraging bagging method, that uses randomization on the weights of the instances of the input stream, to improve the accuracy of the ensemble classifier. Using random output codes, we may use a binary classifier without losing accuracy. We tested subagging, half subagging, and bagging without replacement, and these methods performed faster but they are marginally less accurate. Finally, we have compared our method with Random Forests with improved results.

References

1. Abdulsalam, H., Skillicorn, D.B., Martin, P.: Streaming random forests. In: IDEAS 2007: Proceedings of the 11th International Database Engineering and Applications Symposium, Washington, DC, USA, pp. 225–232. IEEE Computer Society, Los Alamitos (2007)
2. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
3. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: Jonker, W., Petković, M. (eds.) SDM 2007. LNCS, vol. 4721. Springer, Heidelberg (2007)
4. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. Journal of Machine Learning Research, JMLR (2010), `http://moa.cs.waikato.ac.nz/`
5. Bifet, A., Holmes, G., Pfahringer, B., Frank, E.: Fast perceptron decision tree learning from evolving data streams. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) Advances in Knowledge Discovery and Data Mining. LNCS, vol. 6119, pp. 299–310. Springer, Heidelberg (2010)
6. Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavaldà, R.: New ensemble methods for evolving data streams. In: KDD, pp. 139–148 (2009)
7. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
8. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
9. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth, Belmont (1984)
10. Bühlmann, P., Yu, B.: Analyzing bagging. Annals of Statistics (2003)
11. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research 7, 1–30 (2006)
12. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. J. Artif. Intell. Res. (JAIR) 2, 263–286 (1995)
13. Domingos, P.: Why does bagging work? A bayesian account and its implications. In: KDD, pp. 155–158 (1997)
14. Domingos, P., Hulten, G.: Mining high-speed data streams. In: KDD, pp. 71–80 (2000)
15. Friedman, J., Hall, P.: On bagging and nonlinear estimation. Technical report, Stanford University (1999)
16. Friedman, J.H.: On bias, variance, 0/1—loss, and the curse-of-dimensionality. Data Min. Knowl. Discov. 1(1), 55–77 (1997)
17. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 286–295. Springer, Heidelberg (2004)

18. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: KDD, pp. 523–528 (2003)
19. Grandvalet, Y.: Bagging equalizes influence. Machine Learning 55(3), 251–270 (2004)
20. Harries, M.: Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales (1999)
21. Holmes, G., Kirkby, R., Pfahringer, B.: Stress-testing Hoeffding trees. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 495–502. Springer, Heidelberg (2005)
22. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: KDD, pp. 97–106 (2001)
23. Lee, H.K.H., Clyde, M.A.: Lossless online bayesian bagging. J. Mach. Learn. Res. 5, 143–151 (2004)
24. Oza, N., Russell, S.: Online bagging and boosting. In: Artificial Intelligence and Statistics 2001, pp. 105–112. Morgan Kaufmann, San Francisco (2001)
25. Oza, N.C., Russell, S.J.: Experimental comparisons of online and batch versions of bagging and boosting. In: KDD, pp. 359–364 (2001)
26. Saffari, A., Leistner, C., Santner, J., Godec, M., Bischof, H.: On-line random forests. In: 3rd IEEE - ICCV Workshop on On-line Learning for Computer Vision (2009)
27. Schapire, R.E.: Using output codes to boost multiclass learning problems. In: ICML 1997: Proceedings of the Fourteenth International Conference on Machine Learning, pp. 313–321. Morgan Kaufmann Publishers Inc., San Francisco (1997)
28. Street, W.N., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. In: KDD, pp. 377–382 (2001)