Classification Using Streaming Random Forests

Hanady Abdulsalam, David B. Skillicorn, *Member*, *IEEE*, and
Patrick Martin, *Member*, *IEEE Computer Society*

Abstract—We consider the problem of data stream classification, where the data arrive in a conceptually infinite stream, and the opportunity to examine each record is brief. We introduce a stream classification algorithm that is online, running in amortized $\mathcal{O}(1)$ time, able to handle intermittent arrival of labeled records, and able to adjust its parameters to respond to changing class boundaries ("concept drift") in the data stream. In addition, when blocks of labeled data are short, the algorithm is able to judge internally whether the quality of models updated from them is good enough for deployment on unlabeled records, or whether further labeled records are required. Unlike most proposed stream-classification algorithms, multiple target classes can be handled. Experimental results on real and synthetic data show that accuracy is comparable to a conventional classification algorithm that sees all of the data at once and is able to make multiple passes over it.

Index Terms—Data stream mining, data stream classification, decision tree ensembles, random forests.

1 Introduction

Many recent applications such as telecommunication data management, financial applications, sensor data analysis, and Web logs, deal with *data streams*, conceptually endless sequences of data records, often arriving at high rates. Analyzing or mining data streams raises several new issues compared to standard data mining algorithms. Standard data mining algorithms assume that records can be examined multiple times. Data stream mining algorithms, on the other hand, are more challenging to design since they must be able to extract all necessary information from records with only one, or perhaps a few, passes over the data.

Data stream mining algorithms must, therefore, be online, so that each incoming record can be examined in amortized $\mathcal{O}(1)$ time. Arrival rates for records are high, so the practical complexity of processing must also be low. Since the classification algorithm executes endlessly, it must be able to adapt the classification model to changes in the data stream, in particular to changes in the boundaries between classes ("concept drift"). To maximize usefulness, classification should be possible as soon as a sufficiently robust model has been built. Periods when the model is updated because of a change in the underlying data should be as brief as possible. Hence, the classification algorithm should be incremental, so that changes require a model update rather than a completely new model.

Mainstream classification algorithms consist of three phases; a training phase using labeled records, a test phase using previously unseen labeled records, and a deployment phase that classifies unlabeled records. Stream classification algorithms, in contrast, deal with only a single stream of data, in which labeled and unlabeled records are mixed together in the stream. The training/test and deployment phases, therefore, must be interleaved. Stream classification of unlabeled records could be required from the beginning of the stream; after some sufficiently long initial sequence of labeled records; or at specific moments in time, or for a specific block of records selected by an external analyst.

Depending on the distribution of labeled records in the stream, we define some possible scenarios, shown in Fig. 1.

The shaded areas represent the labeled records. Scenario 0 is the basic scenario. Labeled records occur only at the beginning of the stream and there are enough of them to build and test the classification model. Once the model is built, it is used to classify unlabeled records. Clearly, such streams do not permit class boundary changes since the rest of the stream contains only unlabeled records. The only new issue for Scenario 0 algorithms is that they must be fast enough to handle the arrival rates of records.

Scenario 1 introduces the concept drift problem. In this scenario, labeled data blocks are long enough to build or update a model. Each new block of labeled data represents the current state of the system being modeled. The boundaries between the classes in a newly arrived block maybe different from those of previous blocks, and so the classification model must be updated. However, the updating process is straightforward, because the number of labeled records is sufficient to enable a complete update to be carried out.

Scenarios 2 and 3 are versions in which not enough labeled records arrive at a time to guarantee that the classification model can be updated as necessary to reflect the changed class boundaries. Scenario 3 is the most realistic since the labeled blocks have random sizes and frequencies in the stream. For these scenarios, the model update should be able to alter the current model incrementally using any available labeled data, and guarantee that the new model is at least a better approximation than the

- *H. Abdulsalam is with the Information Science Department, College for Women, Kuwait University, PO Box 5969, Safat 13060, Kuwait. E-mail: hanady.abdulsalam@ku.edu.kw.*
- *D.B. Skillicorn and P. Martin are with the School of Computing, Queen's University, Kingston, Ontario, Canada K7L 3N6. E-mail: {skill, martin}@cs.queensu.ca.*

Manuscript received 7 Aug. 2008; revised 8 July 2009; accepted 24 Sept. 2009; published online 9 Feb. 2010.
Recommended for acceptance by D. Cook.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2008-08-0409.
Digital Object Identifier no. 10.1109/TKDE.2010.36.

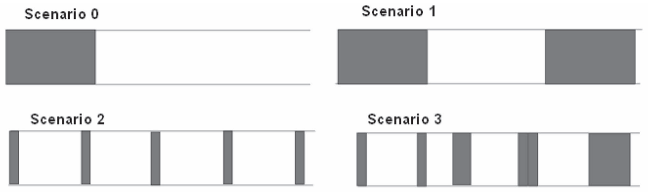

Fig. 1. Possible scenarios for the positions of labeled records in streams.

old one, even if it may not have enough information to fully reflect the changing class boundaries. If the model is unable to classify unlabeled records robustly, it should be able to detect this, and either not classify them, or label the classifications as potentially unreliable. Scenario 3 is the most general; all of the other scenarios are special cases.

The intermittent class labeling of some records is characteristic of a number of real-world systems, where the information used to generate the class labels is only available with some delay. For example, consider a classification system for spam detection. An incoming stream of e-mails is initially unlabeled but, as the user interacts with them, some of their properties can be inferred. The user may label some explicitly as spam, or may never open some of them. On the other hand, replying to an e-mail is a strong signal that it is not spam. These class labels about some of the incoming e-mails can be used to update a spam prediction model that will perform better on newer incoming e-mails. The boundaries between classes will change as the user decides, for example, that some mailing-list e-mails should now be classified as spam. Fig. 2 illustrates the e-mail spam detection system model.

Another example of stream classification are streaming music stations such as Pandora [1]. A classification model tries to predict two classes: “liked songs” and “unliked songs.” As new songs are delivered to the user, there is an opportunity to label them as “liked” or “disliked,” but such labeling tends to be erratic and intermittent. For example, the user may pay attention to the first few songs delivered in each session, but may stop when the delivered songs are “good enough.” A user may leave the computer for a while, but leave the stream playing. The net result is that only some songs are labeled. This example can also be extended to a multiclass classification problem when the users are asked to rank the songs according to their preferences. The higher ranked songs are, therefore, played more often than the lower ranked ones.

The field of data stream mining has attracted a great deal of attention, with research in, for example, detecting changes in data streams [2], [3], [4], maintaining statistics of data streams [5], [6], finding patterns in data streams [7], [8], data stream classification [9], [10], [11], [12], [13], [14], [15], [16], [17], and data stream clustering [18], [19].

Despite the problems already addressed in the field of data stream classification, there are still limitations in the proposed algorithms. Some of the most important ones are:

- Many current stream classification algorithms are designed and/or tested only on two-class classification problems [9], [10], [11], [12], [13], [14], [15], [20].
- Classifiers in current algorithms require extremely large amounts of data for training. Domingos and Hulten [9], for example, demonstrate that their streaming decision tree algorithm (VFDT) greatly improve its accuracy after it sees around 1 million records for constructing only one tree. Much of the research work for streaming algorithms justify using huge number of data records by arguing that data streams are endless, and therefore, the amount of data used is not an issue. We believe that algorithms should, however, use as little data for training as possible to minimize the build and update times for classifiers.
- Many current stream classification algorithms have classification accuracy that is worse than a conventional algorithm would achieve on (a subset of) the data stream; and cannot handle changes in the class boundaries caused by *concept drift*.
- Many ensemble stream classification algorithms are based on batch learning [11], [12], [13], [14], [15], [20], [21], [22], which prevents them from being fast and incremental.

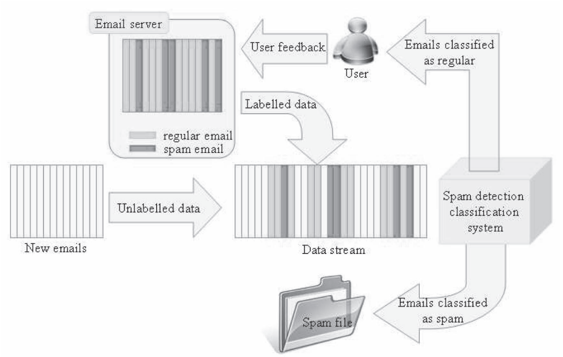

Fig. 2. E-mail spam detection system model.

The main contribution of this paper is to introduce an incremental stream classification algorithm that achieves high classification accuracy, even for multiclass classification problems. The algorithm combines the ideas of streaming decision trees and Random Forests. The algorithm is intricate, so we introduce its explanation in three phases. The first phase (Section 3) [23] introduces the way in which the streaming decision tree construction is merged with the Random Forests algorithm to formulate an incremental stream classification algorithm that handles Scenario 0 with a classification accuracy comparable to (or better than) standard classification algorithms. The second phase (Section 4) [24] shows how the algorithm can be extended to handle concept drift in the stream, that is new labeled records that imply a different set of decision boundaries among the classes. The algorithm of this phase is a self-adjusting algorithm that handles Scenario 1 using an *entropy-based change-detection technique*, which we define based on ideas proposed by Vorburger and Bernstein [3]. The third phase (Section 5) introduces our new contributions for handling Scenarios 2 and 3. The key feature of this phase is that the algorithm is now able to decide whether the current model is ready for deployment or not, when the number of labeled records is not enough to completely build/update the model. A significant contribution of this version is, therefore, the definition of a threshold value on which the decision to deploy the current model is based. We evaluate the phases on both synthetic and real data having blocks of labeled records with random arrival rates and random sizes.

Our algorithm handles only numerical or ordinal attributes for which the maximum and minimum values of each attribute are known. It also assumes that the records are approximately uniformly distributed across all classes, although a known distribution for classes could be included straightforwardly.

2 Background

We assume that the reader is familiar with the standard algorithms for constructing decision trees [25]. Here, we review how a decision tree can be built from stream data, how entropy measures have been used to detect concept drifts, and the design of the Random Forests ensemble algorithm.

2.1 Streaming Decision Trees

A decision tree can be constructed from labeled stream data, using a construction due to Domingos and Hulton [9]. A decision tree being constructed consists of *internal nodes*, each of which contains an inequality on one of the attributes, *frontier nodes*, which are nodes that have not yet been either split or turned into leaves, and *leaf* nodes. Initially, a tree consists of a single frontier node. As each record arrives, it is routed down the tree, based on its attribute values and the inequalities of the internal nodes, until it reaches a frontier node, where it becomes part of the data that will be used to decide on the best attribute and split point for that node.

The Hoeffding bound [26] is used to decide when a frontier node has accumulated enough records for a robust decision about its attribute test to be made. The Hoeffding bound states that, given a random variable r in the range L, and n independent values of r having mean $\overline{r}$, the true mean of r is at least $\overline{r} - \epsilon$, where

$$\epsilon = \sqrt{\frac{L^2 \ln(1/\delta)}{2n}}$$

with probability $1 - \delta$, and δ is a user-defined threshold.

The Hoeffding bound is used in the following way. Assume that we have a general function G that checks an attribute's goodness for splitting, such as information gain [27] or Gini index [25]. At each frontier node, G is calculated for all attributes and the best and second best attributes are used to calculate $\Delta G = G_{highest} - G_{second\_highest}$.

The algorithm recalculates G for all attributes as each new record arrives, or for every small number of data records (n_{min}), and updates ΔG continuously until it satisfies a stopping condition, $\Delta G > \epsilon$. At this point, the true value of the largest G is within ϵ of the approximated G with probability $1 - \delta$. The attribute with the highest G is the best choice for splitting at the current node with confidence $1 - \delta$.

A node is assigned to be a leaf node only if the records that have arrived at this node, by the end of the training block, are all from the same class. Clearly, this method of constructing streaming decision trees is able to only handle up to Scenario 1.

2.2 Using Entropy to Detect Concept Drift

Shannon's entropy [28] is a measure of the disorder associated with a random variable. It is defined as

$$H(x) = -\sum_i p_i \log_2(p_i),$$

where x is a discrete random variable, and p_i is the probability of occurrence of x_i. Finding the entropy for a data set with known distribution requires using the following equation:

$$H(x) = -\sum_x P(x) \log_2(P(x)),$$

where $P(x)$ is the probability mass function of x. In order to use entropy in the context of detecting concept changes in data streams, a two-window paradigm [2], [3] is typically used. Two sliding windows over the data stream are tracked; one is the current window of data, and the other is a reference window that remembers the distribution of the stream. Algorithms for detecting concept changes compare the entropies of the current and reference windows; if the entropies differ by more than a defined amount then a change is deemed to have occurred.

2.3 The Standard Random Forests Algorithm

The Random Forests algorithm is an ensemble classification technique developed by Breiman [29]. As with any tree ensemble classifier, it grows a number of binary decision trees and predicts the class of each new record using the plurality of the class predictions from the set of trees. However, it differs from standard ensemble techniques in the way in which records are selected to grow each tree, and the way in which attributes are selected at each internal node.

Suppose that a data set contains n records, each with m attributes. Each tree is grown by:

- Choosing a subset of the n records, at random with replacement, to form a training set.
- For each internal node, a subset of M ($M \ll m$) randomly chosen attributes are selected, and the decision about which attribute and split point is made using the standard Gini index algorithm on only the selected attributes. A typical value of M (suggested by Breiman) is $\log_2 m + 1$.
- The tree is left unpruned.

The random selection of records with replacement leaves some records (about a third of them) that are never used in the building of this tree. These records can be used to estimate the error rate of each tree, even as it is being built.

This process is repeated with a fresh subset of the records to produce a user-specified number of trees. The choice of attribute and split point at each internal node of each tree is made in a doubly contextualized way—it depends on the other records that were chosen to build *this* tree, and on the other attributes that were chosen to build *this* internal node. This property makes overlearning impossible and the predictions of the ensemble extremely robust.

The Random Forests classification error depends on:

- The correlation among trees: the smaller the correlation among the trees the more variance canceling takes place;
- The strength of each individual tree: the more accurate each tree is, the better its individual vote.

3 The Basic Streaming Random Forests Algorithm

The basic idea is to merge the techniques used for building decision trees from streams, and for building Random Forests to produce an efficient and accurate algorithm, that can be updated in an incremental way because the classification model is an ensemble.

The basic Streaming Random Forests algorithm [23] has classification accuracies approximately equal to those of the standard Random Forests algorithm, but uses many fewer records for training than comparable stream classification algorithms [9], [10].

The standard Random Forests algorithm repeatedly extracts subsets of the records, and each one is used to build one of the trees of the ensemble. In a stream, a different process must be used to select records. This is done by selecting a block from the incoming stream. As long as the stream is locally randomly ordered, this selection is equivalent to selection with replacement from a static data set (and if the stream is too ordered, an equivalent effect can be achieved by maintaining a finite-length buffering, and selecting with replacement from the buffer). The size of the block selected is $tree\ window$. The standard Random Forests algorithms also uses the unselected records as a test set. In the streaming version, this is done by selecting a block of the appropriate size to act as the test set. Again, if the stream is sufficiently randomly ordered, this is equivalent to the standard case.

Once a set (block) of records has been selected, the Random Forests tree-building algorithm is followed, with some refinements. Each newly arrived record is routed down the tree under construction, based on its attribute values, until it reaches a frontier node, where the attribute values of the record are used to calculate $class\ counts$ that contribute to computing the Gini indexes. The values of each attribute (numerical or ordinal) are discretized into fixed-length intervals. The boundaries between the intervals become the possible split points.

Every time a frontier node has seen a block of records of user-defined size n_{min}, both the Hoeffding and Gini tests are applied. If the Hoeffding bound test is satisfied, then the frontier node is transformed into an internal node with an inequality based on the best attribute and split point given by the Gini index tests. The two children of this node become new frontier nodes.

If the number of records that have reached the frontier node exceeds a threshold, $node\ window$, and the node has still not been split, the algorithm transforms the node into a leaf, if the accumulated records are almost all from one class. If not, the node is transformed to an internal node based on the best attribute and split point so far. A more detailed explanation of the tree-building procedure and the tree-pruning method can be found in Abdulsalam et al. [23].

4 Extending the Algorithm to Handle Concept Drift

We now explain how to extend the algorithm to handle concept changes in the incoming stream (Scenario 1). We also add features that enable the algorithm to tune its own parameters in response to the data that it sees, relieving the analyst of the need to carefully define algorithm parameters.

The dynamic algorithm builds a user-defined number of trees. However, while $tree\ window$ was fixed in the basic algorithm, it now changes depending on properties of the stream. The basic tree-building strategy is as before. Incoming records pass down the tree to frontier nodes, where the Hoeffding and Gini tests are applied each time n_{min} records have reached them.

The classification error of each tree is calculated after it has seen a total of $tree_{min}$ records, using a test set that contains 10 percent of $tree_{min}$ unseen labeled records. This percentage is obtained empirically. As any classification algorithm, test records are classified using the built tree, and classification errors are then calculated by finding the percentage of misclassified records to the total number of records in the test set. If the tree error is less than a threshold ($tree\ threshold$), then the current tree is considered complete. If the tree error is still too large, the algorithm resumes building the current tree but with $tree_{min}$ set to half its previous value. Each tree, therefore, consumes a different number of labeled records in its construction, but never exceeding $2 * tree_{min}$ records. The parameters n_{min}, $tree_{min}$, and $tree\ threshold$ are assigned values initially.

When the algorithm has grown the required number of trees, it enters a test phase that calculates the classification error of the entire forest, as well as the classification error for each individual tree i. The individual errors ($tree_{error_i}$) are used to assign a weight to each tree's prediction. In addition, the algorithm derives new values for the parameters $tree\ threshold$, n_{min}, and $tree_{min}$ to use in the next building phase (when a new block of labeled records arrives at time $t + 1$) as follows:

- $tree\ threshold(t+1) = \frac{1}{U}\sum_{i=1}^{U} tree_{error_i}(t)$, where U is the number of trees in the forest.
- $n_{min}(t+1) = \frac{ln(1/\delta)}{2(\overline{\Delta Gini(t)})^2}$, where
$$\Delta Gini(t) = Gini_{highest} - Gini_{second\_highest}$$
is computed during the building phase at time t, and $\overline{\Delta Gini(t)}$ is the average value of $\Delta Gini(t)$.
- $tree_{min}(t+1) = n_{min}(t+1) * \overline{tree_{size}(t)}$, where $\overline{tree_{size}(t)}$ is the average of all tree sizes from the building phase at time t, measured in number of nodes per tree.

When a new block of labeled records arrives, a test must be made to see whether the class boundaries it implies have changed. If they have not, 25 percent of the trees are replaced. (This could probably be reduced substantially, but it guarantees that even small concept drifts that might not be detected are taken into account.) If the class boundaries have changed, a greater percentage of the trees are replaced. In both cases, the trees that are replaced are those with the largest values of $tree_{error}$. If no change in class boundaries is detected, the parameters from the previous learning phases are used; if a change is detected, these parameters are reset to their initial values since they were tuned based on the previous concept of the data.

There are three categories of concept drift:

- *Category 1:* Boundaries appear to change because noise in the data creates records that seem to be on the "wrong" side of the existing boundaries. There is no real change in the underlying reality.
- *Category 2:* There is a gradual change in the underlying reality. Therefore, the number of records that match the old boundaries decreases, whereas the number of records from the new reality gradually increases, until all records in the stream represents the new underlying reality.
- *Category 3:* There is a sudden change in the underlying reality, and most records seem suddenly to be on the "wrong" side of boundaries.

These concept drifts are detected using an entropy-based change-detection technique, based on the two-windows paradigm described in Section 2.2. The difference in entropy for the current and reference windows is calculated for each attribute using counters to find the probabilities of occurrences for each value of the attribute. The differences are then averaged to compute the average change in entropy.

The average entropy difference (H) is normalized by dividing it by its maximum possible value: $H' = H/|\log_2 1/C|$. The value of the normalized entropy difference H' is hence in the range [0, 1] and represents the percentage change in the underlying concept. A threshold value that depends on the accumulated average of the computed entropies since the last change is defined. The algorithm records a change in distribution if $H' > \gamma + H_{AVG}$, where H_{AVG} is the accumulated average of the entropies differences computed since the last recorded change, and γ is a constant threshold defined empirically.

The number of trees to replace depends on both the presence of a concept change and the magnitude of the change. The greater the difference between entropy values, the greater the percentage of trees to replace. Since H' is in the range [0, 1], it is used to find the percentage of the number of trees to replace, R, calculated as

$$R = \begin{cases} H' * U, & if(H' * U + \frac{1}{C}((1 - H')U) > U/2), \\ H' * U + U/2, & otherwise, \end{cases}$$

where U is the total number of trees. This equation considers H' as the fraction of the total number of trees to replace only if the remaining set of unchanged trees, $(1 - H')U$, has a higher probability of contributing to making the correct classification prediction when combined with the replaced set of trees. If R is less than 25 percent of the number of trees U, then it is set to $R = R + 0.25 * U$. If the number of trees to be replaced plus the number of remaining trees that are expected to make a correct classification is less than half of the total number of trees, then the number of trees to replace is $H' * U + U/2$.

If two concept drifts occur close to each other, so that the algorithm detects another concept drift before it completes replacing the R trees, then it does not record a further change until it finishes its current update phase. This ensures that the algorithm does not go into an infinite number of update phases when many small concept drifts appear close to each other. The algorithm's behavior when a new block of labeled records arrives is shown in Fig. 3.

```
/*To update the forest at time $t+1$*/
 compute normalized entropy ($H'$)
 compute $H_{AVG}$
 if $H' > \gamma + H_{AVG}$
   call BuildTree to replace the least
    accurate $w\%$ of trees using
    $tree\,threshold(t+1)$, $n_{min}(t+1)$,
    and $tree_{min}(t+1)$
 else
   record a change in distribution
   reset $tree\,threshold$, $n_{min}$, and
    $tree_{min}$ to initial values
   calculate $R$
   call BuildTree to replace the least
    accurate $R$ trees
 end if
```

Fig. 3. Algorithm's behavior when a new block of labeled data arrives.

5 Extending the Algorithm to Handle Random Blocks of Labeled Data

In Scenarios 2 and 3, a block of labeled records may not be long enough to grow the required number of trees for updating the forest. The block maybe as small as a single labeled record. We now describe the extension to the algorithm to enable it to update its model using blocks of labeled records of any length; and then determine whether the resulting classifier is "good enough" to be deployed, or whether the algorithm should wait for further labeled records and use them to improve its quality first. This evaluation must be made whenever a block of labeled records ends. We call this phase the *evaluation phase*.

The criterion that the algorithm uses during the evaluation phase is based on the *margin function* (mg) defined by Breiman [29]. A set of labeled records, called the *evaluation set*, is withheld from the labeled records encountered since the preceding evaluation phase. One percent of the labeled records are stored, we call the number of records $eval\ cnt$.

5.1 The Margin Function

The margin function was defined by Breiman [29] for Random Forests to measure the strength of an ensemble of classifiers during construction. It measures how far the average number of correct votes for a specific record is from the average number of (incorrect) votes of the next most popular candidate class. Given a Random Forest with U trees $tree_1, tree_2, \ldots, tree_U$, the margin function is defined to be

$$mg(X, Y) = avg_U I(tree_u(X) = Y) \\ - max_{j \neq Y} avg_U I(tree_u(X) = j),$$

where $1 \leq u \leq U$, $I(.)$ is the indicator function, X is a labeled record from class Y, and $tree_u(X)$ is the class voted for by $tree_u$ for record X. Clearly, the larger the value of mg, the stronger the forest.

5.2 Initial Forest-Building Phase

Initially, the algorithm attempts to grow the defined number of trees based on the initial block of labeled records in the stream. If the number of labeled records in the initial block is not sufficient to completely build the required forest, the algorithm enters an evaluation phase, but only if enough trees were grown for evaluation. The threshold of the number of trees that must have been built before a first evaluation phase is 10 percent of the total required number of trees.

In the evaluation phase, the average of the margin value, $\overline{mg}$, for the current forest is calculated, ignoring any tree that might be partially constructed when the labeled records end. We define $\overline{mg}$ to be

$$\overline{mg} = \frac{1}{eval\ cnt} \sum_{i=1}^{eval\ cnt} mg_i(X_{\subset \text{ evaluation set}}, Y).$$

If the calculated value of $\overline{mg}$ is greater than a threshold, $mg_{threshold}$, then the forest is seemed to be robust enough, and it is deployed to classify unlabeled records. In this case, the algorithm clears the evaluation set, enters a test phase to enable the measurement of what the deployed classification error would be (needed to evaluate the algorithm), and starts collecting new evaluation records when incoming labeled records appear. If $\overline{mg}$ is not greater than $mg_{threshold}$, then no classification of unlabeled records takes place, and the algorithm waits for the next block of labeled records to resume building the forest to the specified size.

5.3 Deciding When the Forest is Good Enough

A simple way to define the value of $mg_{threshold}$ is that it is the number of correct votes for a class if the forest was to vote uniformly randomly. This is the ratio of the current number of trees in the forest under construction, U, to the number of classes of the data set, C; $mg_{threshold} = U/C$. One argument for choosing this ratio as the threshold is that having more trees in the forest should naturally lead to greater differences between correct and incorrect voting and, hence, larger values of $\overline{mg}$. On the other hand, a data set with more classes results in more possibilities for each tree's vote, and, therefore, smaller differences between correct and incorrect voting and hence smaller values of $\overline{mg}$.

Defining $mg_{threshold} = U/C$ means that the algorithm does not consider a forest accurate enough to deploy unless the number of votes for the correct class exceeds the number of votes produced by uniform voting (U/C) by at least $v = \frac{C-1}{C^2}U$, in the worst case where all the incorrect votes are uniformly distributed across the other $C-1$ classes. We derive the minimum increment of the number of correct votes (v) from the following equation, which describes a forest with uniformly random voting:

$$v + U/C - (U/C - v/(C-1)) \geq (mg_{threshold} = U/C).$$

The equation assumes that v is taken equally from all other classes, hence, the incorrect votes are distributed evenly among the other $C-1$ classes.

Positive values of $\overline{mg}$ define how confident the forest is in its classification. We define the algorithm confidence, $conf$, which will be used to decide when to deploy the ensemble on unlabeled records, by relating $\overline{mg}$ and U linearly, so that

$$conf = \begin{cases} \overline{mg}/U, & \text{if } \overline{mg} \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

In practice, however, this confidence value is quite conservative, and the forest confidence might be greater than what a linear relation suggests.

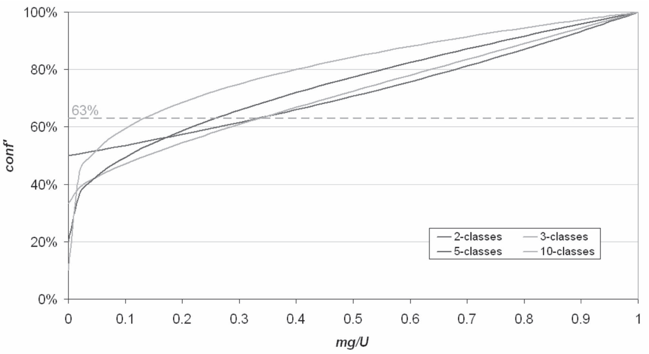

Fig. 4. $conf'$ plots for $C = 2$, 3, 5, and 10.

We, therefore, suggest another definition for the algorithm confidence $conf'$, in terms of $\overline{mg}$, the number of classes C, and the number of trees in the current forest U, such that the following conditions are met:

- If $\overline{mg}$ is negative, then the forest confidence must be zero.
- If the value of $\overline{mg}$ is equal to 0, which means that the forest has voted equally for all classes (uniform random classifier), then the forest must have confidence $\frac{1}{C}$ percent.
- If the value of $\overline{mg}$ is equal to number of trees, U, which means all the trees have voted for the correct class, then the forest must have 100 percent confidence.
- For a constant value of C, the greater the ratio of $\overline{mg}$ to U, the greater the confidence should be.
- For a constant value of $\overline{mg}/U$, the greater the value of C, the greater the confidence of the algorithm should be.
- The confidence function increases monotonically as the value of $\overline{mg}/U$ increases, and the rate of increase should be greater for larger numbers of classes.

A plausible definition of $conf'$ that satisfies these conditions is

$$conf' = \begin{cases} \left(\frac{1}{C}\right)^{\left(1-(\frac{\overline{mg}}{U})^{\frac{1}{C-1}}\right)}, & \text{if } \overline{mg} \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

Fig. 4 shows the plots of $conf'$ versus positive values of $\overline{mg}/U$ for $U = 50$ trees and number of classes $C = 2, 3, 5$, and 10.

Starting from a confidence value of about 63 percent, the function performs as expected. The plots are not necessarily in the correct order in the region below this. This is expected because the function starts at lower confidence values for larger number of classes. This region can safely be ignored, since it is highly unlikely that an acceptable confidence level for a classification application would be below 63 percent.

Another definition of $mg_{threshold}$ is required when using the $conf'$ equation as the algorithm confidence measure. We, therefore, derive $mg'_{threshold}$ from the equation of $conf'$, given the accepted confidence level that is expected from the algorithm, $conf'_{given}$ (perhaps defined by an analyst). The value of $mg'_{threshold}$ is defined as

$$mg'_{threshold} = U * \left(1 - \log_{\frac{1}{C}} conf'_{given}\right)^{C-1}.$$

5.4 Subsequent Building Phases

Except for the building phase using the very first block of labeled data, three conditions may occur when a new block of labeled records is encountered, and each implies that a different number of target trees remain to be grown in the current update phase.

- If the total number of trees is still less than the number of trees required for the forest (that is, the first block of labeled records was too small to build a complete forest), then the target number of trees is the remaining number of trees to complete the forest for the first time.
- If the total required number of trees has been grown, and there is no concept drift, then the target number of trees is 25 percent of the total number of trees, as before.
- If the total required number of trees has been grown, and there is some concept drift, then the target number of trees to replace is R, as before (see Section 4).

The algorithm grows the target number of trees one by one. Each time the algorithm completes a tree, it either adds the tree to the forest if the total number of trees has not yet been reached, or it replaces the existing tree with the worst classification performance by the new tree.

When the current block of labeled data ends, there are two different situations that must be dealt with.

- The required number of trees to build has been reached (that is, the block of labeled records was long enough). In this case, the algorithm acts like the dynamic version and enters a test phase to record the classification accuracy.
- The target number of trees has not been reached (that is, the block of labeled records was too small). In this case, the algorithm enters an evaluation phase only if two conditions are satisfied:
    - The number of trees already grown is at least 10 percent of the total number of trees.
    - The algorithm has at least seen $tree_{min}$ records since the previous evaluation phase, when there is no concept drift, or since the last detected concept drift. This ensures that at least one additional tree has been built since the most recent event (evaluation phase or concept drift detection). The relation $eval\ cnt \geq 0.01 * tree_{min}$ must therefore hold.

  Otherwise, the algorithm does not enter an evaluation phase. It waits until the next block of labeled records arrives and resumes the update phase by continuing to build the tree currently under construction and keeps on building trees to reach the target number. While a tree is under construction, it is never included in evaluating the forest.

  As in the initial building phase, when the algorithm completes an evaluation phase and the forest is ready for deployment, the classification error for the evaluation set is calculated, the evaluation set is cleared, and the algorithm enters a test phase to simulate the deployed classification error. The evaluation set is also cleared when a concept change is detected. This ensures that the algorithm evaluates the forest on data with the same data distribution as the data on which the forest is deployed. If the forest is not ready for deployment, the algorithm waits for the arrival of the next block of labeled records, and resumes the update process.

```
/*building/updating phase*/
 while more trees to build
  call BuildTree
   if block of labelled records ends
    if current number of trees ≥ 10% of total
       number of trees and $eval\ cnt \geq 0.01 * tree_{min}$
     call EvaluateForest to calculate $\overline{mg}$
     if $\overline{mg} \geq (mg_{threshold}$ or $mg'_{threshold})$
      ignore tree under construction in case of
      initial building
      or restore old tree in case of update
      use current forest for classification
      reset evaluation set
     end if
    end if
     continue building trees when a new block of
     labelled records arrives
   end if
   if concept drift is detected
    reset evaluation set
   end if
 end while
```

Fig. 5. Algorithm's behavior when a block of labeled records ends.

If a block of labeled records ends during a test phase, there is no harm since the update phase is over, and the algorithm records the classification error for the test records seen so far. It enters a new update phase when the next block of labeled records arrives. The algorithm's behavior for handling Scenario 3 is shown in Fig. 5.

5.5 Time Complexity

The time the algorithm takes to complete a building phase depends on: 1) the time to build each tree, 2) the time to evaluate the forest, and 3) the time to test the forest.

The time to build each tree is the sum of the times for reading the training records, passing them down trees to the nodes, incrementing the counters, performing Hoeffding and Gini tests every n_{min} records, performing the entropy test for every entropy window of records, and testing the tree. This time is obviously bounded by the time for reading the records, passing them down the tree, and testing the tree, because other computations are only increments and simple comparisons. Each tree is trained from a maximum of $2 * tree_{min}$ number of records, and tested using a maximum of $0.1 * 2 * tree_{min}$ records. Assuming the trees are balanced, during training and testing a record needs to pass through a maximum of $\log_2 \overline{tree_{size}} + 1$ levels to a frontier node. The time to build a tree is therefore $(2 * tree_{min} + 0.1 * 2 * tree_{min}) * (\log_2 \overline{tree_{size}} + 1)$, which gives a complexity of $\mathcal{O}(U * tree_{min} * \log_2 \overline{tree_{size}})$ for building U trees.

The time to evaluate the forest depends on reading the records of the evaluation set and passing them down all the trees of the current forest. Since the evaluation set contains a maximum of one percent of the number of records read, the time for evaluating the forest under construction is

$U * 0.01 * 2 * tree_{min} * (\log_2 \overline{tree_{size}} + 1)$, which reduces to a complexity of $\mathcal{O}(U * tree_{min} * \log_2 \overline{tree_{size}})$.

The time to test the forest depends on reading the test records and passing them down all the trees. The number of test records is constant, so the time complexity for testing the forest under construction is $\mathcal{O}(U * \log_2 \overline{tree_{size}})$.

The time complexity of completing one building phase in our algorithm is therefore $\mathcal{O}(U * tree_{min} * \log_2 \overline{tree_{size}})$. The complexity can be further reduced to $\mathcal{O}(U * tree_{min})$, since $\overline{tree_{size}}$ can at most be $(2 * tree_{min})/n_{min}$, so the value of the term $\log_2 \overline{tree_{size}}$ is much smaller than $2 * treemin$.

6 Implementation and Experimental Settings

6.1 Testing Criteria

The main two criteria by which we evaluate the extensions of the algorithm are the algorithm's way of adapting in the case of concept drifts and the decisions that the algorithm takes when a block of labeled data is not long enough to complete a training phase. A training phase could be the initial forest-building phase, or any later update phase of the forest, regardless of the presence of concept changes or not.

We simulate the presence of blocks of labeled records in the stream by randomly choosing a number of *stop points*. Each stop point defines the end of a block of labeled records, and the start of the next block of labeled records. The number of records between any two successive points, therefore, defines the size of the block of labeled records.

To measure the behavior of the algorithm, we record the following:

- The classification errors of the forest when all training blocks are big enough to build/update the forest (Scenario 1).
- The number of trees to replace, R, each time the algorithm enters an update phase to show an example of the algorithm's dynamic behavior of adjusting the parameters. Please refer to Abdulsalam et al. [24] for more experiments of dynamically adjusting the parameters.
- The classification errors of the forest when blocks might not be big enough to complete build/update phases (Scenarios 2 and 3). We simulate the forest classification error when classifying unlabeled data by forcing the forest into a testing phase when it would be deployed after evaluation. The recorded classification errors are, therefore, either the outputs of the ordinary testing phases (when the algorithm completes any build/update phase), or the testing phases that simulate the deployment of the algorithm. (The testing phases used to simulate the deployment phases are not part of the algorithm. They are only used to show what the current forest's behavior would be when deployed.)
- The values of $\overline{mg}$, $mg_{threshold}$, and $mg'_{threshold}$, each time the forest is considered for evaluation. When evaluating the forest based on $mg'_{threshold}$, we set the accepted confidence level for the algorithm, $conf'_{given}$, to the minimum possible, that is 63 percent.
- The classification errors for records of the evaluation sets to support the decisions of the algorithm about deployment. If the algorithm decides that the forest is not ready for deployment because $\overline{mg} <$ ($mg_{threshold}$ or $mg'_{threshold}$), then the corresponding evaluation set classification error should be large. Otherwise, the classification error for the evaluation set should be in the same range as the known noise rate of the data.
- The confidence values, $conf$ and $conf'$, of the algorithm each time the forest is evaluated.

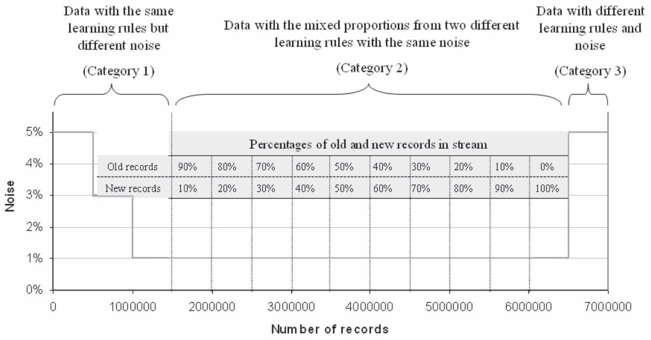

Fig. 6. Concept drift of the synthetic data.

We report results for experiments of Scenarios 2 and 3 twice, one run based on $mg_{threshold}$ and the other based on $mg'_{threshold}$. Both experiments use the same seed, so that they both have the same stop points and use the same splitting attributes for each node. This ensures a fair comparison of the behavior of the algorithm in the two cases. Note that using $mg'_{threshold}$ results in $conf'$ and $\overline{mg'}$.

We also show that the extension to the algorithm does not require much extra time for the forest evaluation at each stop point. We demonstrate that by comparing the time of execution when there are no evaluation phases (streams of Scenario 1) with the time of execution when it passes through evaluation phases (streams of Scenarios 2 and 3).

6.2 Testing Platform

We conduct our experiments on a system with a Pentium 4 3.2 GHz processor and 512 MB RAM. The algorithm is implemented in FORTRAN 77, derived from the code of the Random Forests algorithm [30].

6.3 Data Sets

We test the algorithm on synthetic and real data sets. We use 50 stop points for the synthetic data set, 15 for one real data set, and 10 for the other real data set.

6.3.1 Synthetic Data

The synthetic data sets are generated using the DataGen data generation tool by Melli [31]. The size of the stream is seven million records, containing a range of typical concepts change as shown in Fig. 6. The stream is arranged like this:

- The first block of 1,500,000 records simulates an unchanging underlying stream in which the level of noise is changing. The noise level starts at five percent for the first 500,000 records, then drops to three percent for the next 500,000 records, and then drops further to one percent for another 500,000 records.
- The next 5,000,000 records simulate the second category of concept drift, in which some records reflect a new set of class boundaries, and the percentage of such records is increasing. Each

successive block of 500,000 records has proportions of the data reflecting the old class boundaries, and data reflecting the new class boundaries; and the proportion of "old" data drops by 10 percent in each new block. In other words, the first block of 500,000 records has 90 percent "old" records and 10 percent "new" records, while the next to last block has the percentages reversed, and the last block has data that entirely reflect the new boundaries.

- The last 500,000 records simulate the third category of the concept drift. The records in this block have entirely new class boundaries (so there is a sudden transition) with noise level at five percent.

6.3.2 Real Data

We use two real astronomical data sets, extracted from the Sloan Digital Sky Survey (SDSS)[32]. SDSS is a huge survey that offers astronomical data about more than one quarter of the sky, by capturing optical images. It also provides three-dimensional maps of about a million galaxies and quasars.

Our experiments are based on the sixth release of SDSS, DR6, that was made public in June 2007, and available online at http://www.sdss.org/dr6/. DR6 has more accurate measures than the previous releases of SDSS. The maximum recorded uncertainty for the attributes is about two percent [33].

We extract two subsets, each from a different view of the DR6 database, namely the PhotoObj view and the SpecPhoto view. PhotoObj is a view that contains all the attributes of each photometric object, whereas the SpecPhoto is a view of joined Spectro and Photo objects that have the clean spectra. Each extracted data set contains two classes of astronomical objects, namely galaxies and dwarf-stars for the PhotoObj data set and galaxies and stars for the SpecPhoto data set, and seven numerical attributes. The data sets are described as follows:

- *PhotoObj data set.* The first data set contains five attributes (*u,g,r,i,* and *z*) that represent original color measurements in the photometric color system *ugriz*. The two other attributes in this data set represent the right ascension (*RA*), and the declination (*DEC*) of each object.

  The classes are selected based on the primTarget attribute, which represents the primary target category of the object. The records of the galaxies class (class 1) have a *primTarget* value of TARGET_GALAXY. The records of the dwarf-stars class represent red, white, and brown dwarfs, but all are marked as class 2. These records are selected with *primTarget* values of TARGET_STAR_RED_DWARF, TARGET_STAR_WHITE_DWARF, and TARGET_STAR_BROWN_DWARF.

  The extracted data set contains 500,000 records of class 1 (galaxies), and 41,116 records of class 2 (dwarf-stars). The ratio between the two classes is obviously unbalanced. Approximately 92 percent of the records are of class 1 and only about 8 percent of the records are of class 2. We, therefore, merge the data sets into one data set with balanced class ratios in order to be able to test the algorithm and prove our points. We do this by randomly merging the two data sets by selecting records from class 1 with probability 0.9 and records from class 2 with probability 0.1. Then, we replicate the records from class 2 ten times. The resulting data set contains 782,730 records, with 371,517 records (48 percent) of class 1, and 411,160 records (52 percent) of class 2.
- *SpecPhoto data set.* The second data set contains the same five color attributes but corrected to include the effect of the redshift astronomical measure (*dered_u, dered_g, dered_r, dered_i, dered_z*). It also contains the redshift measure, *z*, the *DEC* attribute, and the *RA* attribute. Records are sorted in descending order according to the *DEC* attribute, which is then removed from the data set. Since the challenge with real data is finding a very large data set with a known concept drift, the SpecPhoto data set is extracted in a way such that we know the point of execution at which a concept drift should be expected. Knowing the center of the galaxy in relation to the Earth, objects were selected with right ascensions in a $90\,\mathrm{deg}$ segment centered toward the galactic center, and at declinations in a $30\,\mathrm{deg}$ cone directly above the galactic plane, and in a $13\,\mathrm{deg}$ band much closer to the galactic plane. (Ideally, we would have liked to sample close to the galactic plane but, for obvious reasons, the sky survey objects are all at greater declinations.)

  We expect that the boundary between galaxies and stars appear quite different for these two sets of objects. First, the closer to the galactic plane the greater the density of dust which obscures, preferentially, the fainter galaxies. Second, the closer to the galactic plane, the greater the density of stars, and the greater the variety of star properties. Hence, sorting the data into a stream where the objects from high declinations come first, followed by the objects with lower declinations should produce a concept drift at the boundary, and we can examine whether and how the classification algorithm handles it.

  The classes of this data set are extracted based on the attribute, *type*, which represents the morphological type classification of the object. It is equal to three for galaxies, and equal to six for stars. The extracted data set contains 65,621 records (72.3 percent) of class 1 (galaxies), and 20,441 records (23.7 percent) of class 2 (stars). We, therefore, replicate the records from class 2 three times, resulting in a data set with percentages of the class 1 and class 2 of 51.7 and 48.3 percent, respectively. We noticed that the data records in the lower range of *DEC* values are not frequent enough to train the classifier after the concept drift is detected. We, therefore, replicate the whole block of data three times to produce a final data set with 158,128 records with the concept drift appearing at record 111,353.

7 Results and Discussion

7.1 Classification Accuracy for Scenario 1

The classification errors in Fig. 7 demonstrate that the algorithm achieves classification accuracies that match those expected from the way the data set was constructed, such that the obtained classification errors are approximately equal to the noise percentage of the data. As the amount of noise in the way the boundaries were constructed decreases,

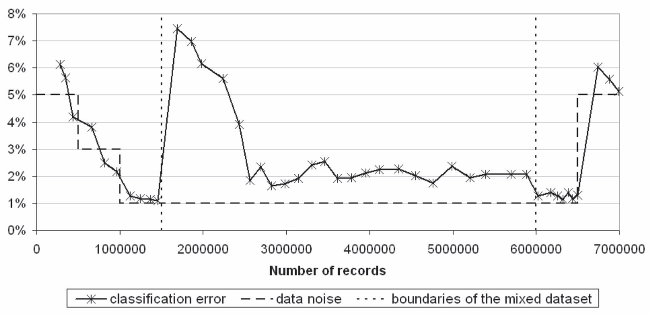

Fig. 7. Classification errors as the algorithm adapts to concepts drifts [24].

the classification error drops quickly. When the mixed part of the stream is encountered, errors climb quickly as the classification model first treats records from the new decision boundaries as noise; but eventually adapts by the time data from the new decision boundaries reaches 30 percent of the records encountered. When data completely from a new set of decision boundaries is encountered (at record 6,000,000), the classification model adapts very quickly.

7.1.1 Comparison with Other Stream Classification Algorithms Based on Classification Accuracy

It is difficult to compare the performance of algorithms from the literature because there are no standard test data sets. However, most implementations have been tested on artificial data sets with known noise, so their performance can be estimated by how close their prediction errors are to the inherent uncertainty in the decision boundaries because of noise. Almost all existing algorithms address only two-class prediction, where only a single decision boundary must be constructed. Multiclass prediction is substantially harder because decision boundaries now intersect, and the areas close to these intersections are much more sensitive to noise in the data and small errors in boundary placement. In this section, we compare the classification accuracy of our algorithm with other stream classification algorithms. Since our algorithm is a decision tree-based ensemble algorithm, we compare it with decision tree-based as well as ensemble-based algorithms.

1. Comparison with decision tree-based stream classification algorithms: We first compare our results with the extensions of streaming decision tree algorithm VFDT [9], namely CVFDT [10] and sVFDT [16]. The CVFDT algorithm has a reported classification error of at least double the error percentage of the synthetic data used (for two classes), after seeing many more data records than our algorithm requires. Tsai et al. [17] present a streaming classification algorithm and provide a comparison with the CVFDT algorithm. Both algorithms have similar accuracies at initial stages, but the Tsai algorithm's accuracy does not drop with increasing amounts of data as the CVFDT algorithm's does. The sVFDT algorithm [16] has a reported classification error close to the data error, but only after seeing 10^7 data records. Existing decision tree algorithms require much more data than our algorithm, and tend to have larger errors. In other words, the use of the random forests construction for each individual tree in our algorithm already improve on previous decision tree construction algorithms—and, of course, these algorithms can only handle concept drift by discarding the current model and learning a new one.

The use of ensembles provides two potential sources of performance improvement: variance canceling by the use of multiple component predictors, and a smooth way to respond to concept drift by replacing some component predictors by newer ones. However, these theoretical improvements have proven difficult to achieve in practice.

2. Comparison with ensemble-based stream classification algorithms: There are a large and growing number of stream classification ensemble algorithms [11], [12], [13], [14], [15]. These use a variety of component conventional classification algorithms in building their predictors (i.e., multiple passes for data batches), and have only been tested on two-class problems.

Wang et al. [11], Masud et al. [12], and Chu and Zaniolo [13] test their tree-based ensemble algorithm on synthetic data with five percent noise and so five percent expected error. Wang et al. [11] report a classification error of about 10 percent for the early part of the stream, but this grows to 18 percent by the time 200,000 records have been processed. The algorithm gives better initial results using Naive Bayesian classifiers, achieving a prediction error of about seven percent for the early part of the stream, but does not report whether or not this increases as more of the stream is processed. Masud et al. [12] report a classification error of at least 10 percent as well. Chu and Zaniolo [13] report an average classification error of about six percent after seeing 320,000 records, on data that includes both sudden and gradual concept drifts.

The SEA and ICEA algorithms [14], [15] are tested on fairly simple three-dimensional synthetic data with two directly related attributes, 10 percent noise, and sudden concept drifts. The reported classification error for both algorithms varies from around 2 to 23 percent after seeing 25,000 records.

Only the Chu and Zaniolo algorithm achieves a prediction error rate comparable to the best achievable, and then only on a two-class problem using conventional predictors. In contrast, our incremental algorithm almost always reaches the expected error rate, even for multiclass data sets. It seems essential both to use strong component predictors, and sophisticated replacement strategies to achieve good performance.

7.2 Dynamic Adjustment of R

Fig. 8 shows that the algorithm initially grows 50 trees, then it replaces 25 percent of the trees (12 trees) when no concept change is detected. In the case of detecting concept change, the number of trees to replace increases based on the significance of the change. For the first six drift points that the algorithm detects, the number of trees to be replace alternates between 29 and 30 trees, which shows that the significance of the changes are almost equal. The number of trees to replace for the last detected drift is 39. This is because this drift is of category 3, which results in sudden change in the underlying reality.

7.3 Margin Function Test

Fig. 9 shows the margin function values $\overline{mg}$ and the margin threshold values $mg_{threshold}$ and $mg'_{threshold}$, for two runs on the synthetic data set. Note that both runs result in the same

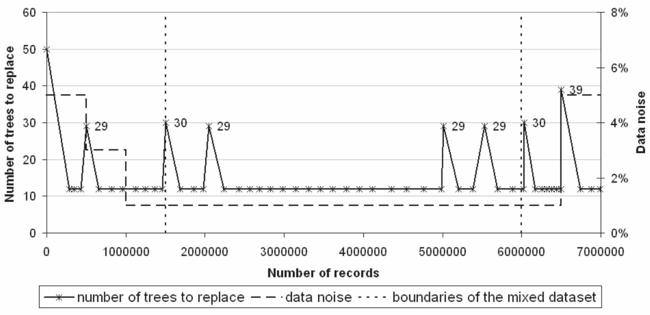

Fig. 8. Number of trees to replace.

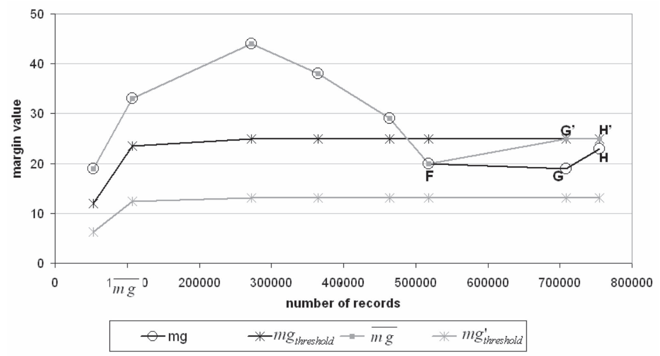

Fig. 10. Values of $\overline{mg}$, $\overline{mg'}$, $mg_{threshold}$, and $mg'_{threshold}$ for PhotoObj.

behavior since no points fell in the gap between the values of $mg_{threshold}$ and $mg'_{threshold}$, and so the decisions of the algorithm for both runs are the same. This values of $\overline{mg}$ and $\overline{mg'}$ coincide exactly and so are plotted as a single graph in the figure. Although the definition of $mg_{threshold}$ has stricter bounds than $mg'_{threshold}$, the graph of $mg_{threshold}$ has lower values. This is because setting $mg'_{threshold}$ to 10 for the case of 50 trees (the value of $mg_{threshold}$) results in a confidence, $conf' < 50\%$, which lies in the ignored area of the $conf'$ graph.

Marks in all plots (asterisks and squares) represent a stop point in the flow of labeled records. Five stop points do not allow evaluation phases: at records 24,872, 1,892,602, 2,439,275, 5,025,766, and 5,074,009, marked as A, B, C, D, and E, respectively. At the first point (A), the forest has not yet completed building 10 percent of the total numbers of trees. The forest was therefore not big enough to be evaluated.

At the points B and C (at records 1,892,602 and 2,439,275), the forest is in test phase and so does not enter an evaluation phase since the forest has been completely updated. The classification error calculated so far in these testing phases is, however, recorded. At the points D and E (at records 5,025,766 and 5,074,009), there is no evaluation phase because the evaluation sets were too small to evaluate the forest.

The figure also shows that, at the remaining stop points, the value of $\overline{mg}$ always exceeds the values of $mg_{threshold}$ and $mg'_{threshold}$, except for the three stop points at records 6,577,271, 6,637,870, and 6,639,975, marked as F, G, and H. At these points, the algorithm, accordingly, reports that the forest is not ready for deployment. Everywhere else, it reports that the forest, though still under construction, is ready for deployment on unlabeled records.

Fig. 10 shows the margin function values, $\overline{mg}$ and $\overline{mg'}$, and their associated threshold values, $mg_{threshold}$ and $mg'_{threshold}$, for the PhotoObj real data set. Seven stop points did not allow the algorithm to enter an evaluation phase because the evaluation sets were not large enough to evaluate the forest. These points are not shown in the graph. The graphs show only the points at which the algorithm enters evaluation phases.

The two runs give the same behavior up to record 518,351, marked as F. At this stage, the margin value $\overline{mg}$ does not exceed its threshold $mg_{threshold}$, whereas $\overline{mg'}$ does exceed $mg'_{threshold}$. The algorithm, therefore, behaves differently starting from this point.

The run that is based on $mg_{threshold}$ does not allow the forest to enter an evaluation phase at point F. In addition, the forest does not enter evaluation phases for the two subsequent points (marked as G and H), because their $\overline{mg}$ values are still less than the value of $mg_{threshold}$. The run based on $mg'_{threshold}$, on the other hand, decides that the forest can be deployed at the execution points F, G', and H', since the values of $\overline{mg'}$ at these points exceed the value of $mg'_{threshold}$. (The labels F, G, G', H, and H' that appear in the later figures for a specific data set represent the same points of execution.)

7.4 Classification Errors for Evaluation Sets

Figs. 11 and 12 show the classification errors for the evaluation sets, for the synthetic and the PhotoObj data sets, respectively. We present these error values to support the decisions that the algorithm takes after evaluating the forest.

Fig. 11 shows that the points at which the algorithm decides that the forest is not ready for deployment, points marked as F, G, and H, have corresponding classification errors of 61.3, 36.7, and 35.0 percent, respectively, when

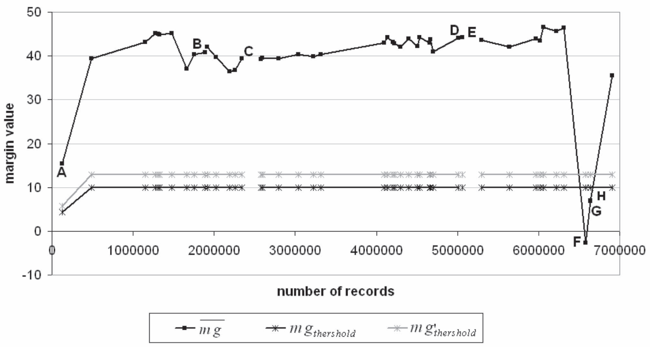

Fig. 9. Values of $\overline{mg}$, $mg_{threshold}$, and $mg'_{threshold}$ for synthetic data.

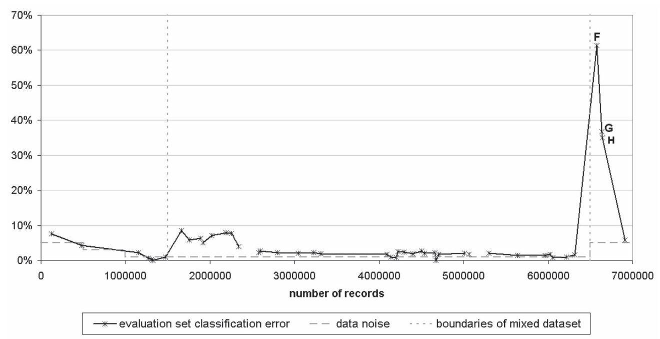

Fig. 11. Evaluation set classification errors for synthetic data.

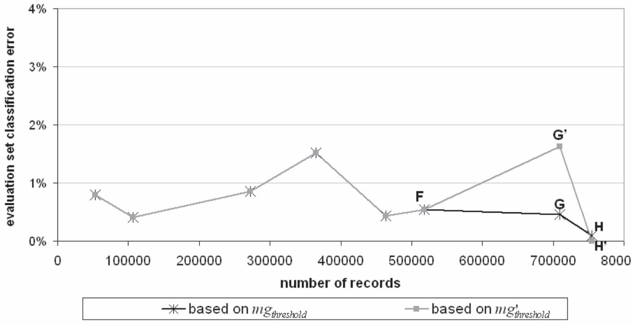

Fig. 12. Evaluation set classification errors for PhotoObj.

noise is only five percent. Obviously, the forest is not ready for deployment at these points since the errors are extremely high compared to what might be expected from the noisiness of the data.

The errors of the remaining stop points, where the algorithm decides that the forest is ready for deployment, are approximately equal to the noise in the data. This demonstrates the appropriateness of the algorithm's decisions. Some points at the start of the period where the category 2 drift is simulated (the period from record 150,000 to record 6,500,000), however, have higher classification errors than expected. The reason is that the early gradual mixing of records from new and old class boundaries causes the algorithm to consider the smaller percentage of records describing new classes as noise.

Fig. 12 shows the evaluation set classification error for the PhotoObj real data set. The behavior of the algorithm for the two runs is the same until it reaches the point F. At this point, and its two subsequent points (G and H), the run that is based on $mg_{threshold}$ decides that the forest is not ready for deployment since $\overline{mg} < mg_{threshold}$ (Fig. 10). The evaluation set classification errors are, however, low—even lower than other points at which the algorithm does decide that the forest is ready for deployment. This shows that $mg_{threshold}$ is too conservative, and its use wastes opportunities to deploy a forest with usable accuracy.

For the run based on $mg'_{threshold}$, F, G$'$, and H$'$ record values in the same range as other points, demonstrating that the decision to deploy the forest at these points is appropriate. Using $mg'_{threshold}$ as the base for the algorithm decisions is more efficient than basing the decisions on $mg_{threshold}$.

7.5 Algorithm Confidence

Fig. 13 presents the confidence values $conf$ and $conf'$ at each evaluation phase for the synthetic data set. The algorithm always has at least 70 percent confidence when deployed on unlabeled records for both confidence values. The points at which the values of $conf$ and $conf'$ are relatively poor (points marked as F, G, and H for $conf$, and F, G$'$, and H$'$ for $conf'$) are the cases where the algorithm decides that the forest is not ready for deployment. One of the three points (point F) has a zero confidence of correct classification because the value of $\overline{mg}$ for this specific point is negative (See Fig. 9).

For the graph that represents $conf$, the points G and H, have confidence values of 13.75 and 13.90 percent, respectively. This shows that the definition of $mg_{threshold}$ guarantees high confidence, since even if the value of $\overline{mg}$ is in the range [0, $mg_{threshold}$] (which means the average of the correct class votes is greater than all of the other classes), the forest might still have poor confidence of correct classification.

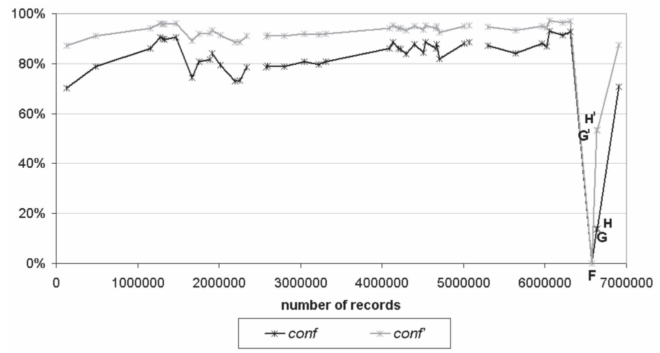

Fig. 13. Confidence values for synthetic data.

For the graph that represents $conf'$, the points G$'$ and H$'$, have confidence values of 53.29 and 53.44 percent, respectively, which are less than $conf'_{given}$ (63 percent). This shows that the measure $mg'_{threshold}$ does not allow the forest to be deployed unless it has confidence values greater than the confidence defined by the analyst.

The definition of $conf'$ is more useful. A uniformly random classifier still has some chance of getting the classifications correct, which should lead to a confidence above zero when $\overline{mg}$ is equal to zero, and correspondingly, greater confidence when $0 < \overline{mg} < mg_{threshold}$. This is only true when calculating the confidence using $conf'$ instead of $conf$.

Fig. 14 shows the confidence values $conf$ and $conf'$ for each evaluation phase for the PhotoObj real data set. The values of $conf$ have a minimum of 58 percent confidence for the points at which the algorithm has decided to deploy the forest. The points marked by F, G, and H are the points at which the algorithm has decided that the forest is not ready for deployment. These points have $conf$ values of 40, 38, and 46 percent, respectively. The values here do not reflect the evaluation set classification errors shown in Fig. 12, since the definition of $conf$ has strict bounds.

The values of $conf'$ have a minimum confidence value of 66 percent, which is more than the value of $conf'_{given}$ (63 percent). This is why the algorithm decides to deploy the forest at all evaluation points, and these points also have good evaluation set classification errors associated with them (see Fig. 12). As for the synthetic data set, the values of $conf'$ are more representative than the values of $conf$ for this data set.

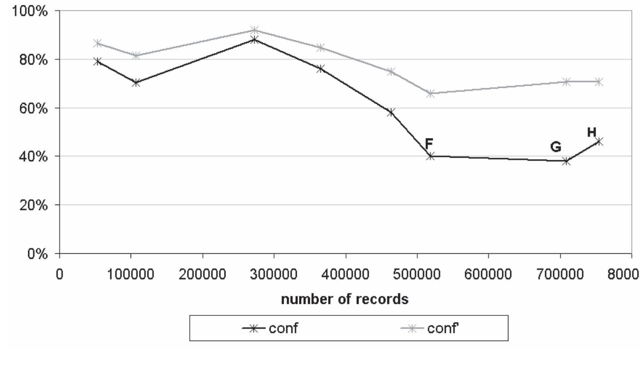

Fig. 14. Confidence values for PhotoObj.

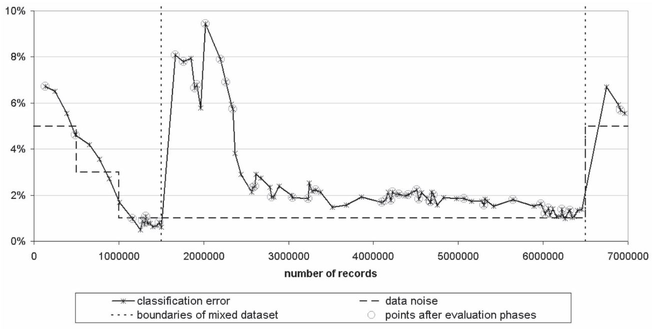

Fig. 15. Forest classification errors for synthetic data.

7.6 Classification Accuracy of the Forest for Scenarios 2 and 3

Figs. 15, 16, and 19 demonstrate the classification error of the forest each time it passes through a testing phase for the synthetic, PhotoObj, and SpecPhoto data sets, respectively. The asterisks surrounded by circles in all figures record the points at which the forest has passed through a test phase after it was evaluated and was ready for deployment (simulated deployment phases). The asterisks surrounded by diamonds in Figs. 16 and 19 are the points of testing the forest after detecting concept drift.

Fig. 15 only plots one graph for the algorithm's behavior for the two runs since, as mentioned earlier, the algorithm decisions for deploying the forest based on $mg_{threshold}$ and $mg'_{threshold}$ are the same. The test phase errors for the simulated deployment phases are approximately equal to the evaluation set classification errors at the points at which the forest is ready for deployment shown previously in Fig. 11. This confirms that the evaluation set classification errors are good indicators of the algorithm's behavior on unlabeled records. The points that are represented only by asterisks are the ordinary testing phases.

Fig. 16 shows that the two runs for the PhotoObj real data set result in the same behavior until the point marked by F (record 483,395). At this point, the two plots start behaving differently because of the different decisions of deploying the forest based on the values of $mg_{threshold}$ and $mg'_{threshold}$.

The two plots reach a maximum 1.8 percent classification error. The error reaches its maximum at the point marked by F, and decreases until it reaches its minimum toward the end of the stream. There is a trade-off in using $mg'_{threshold}$ to decide to deploy the forest. The forest is deployed with large classification errors, but fewer unlabeled records are wasted, since the forest is deployed more often than when basing the decision on $mg_{threshold}$.

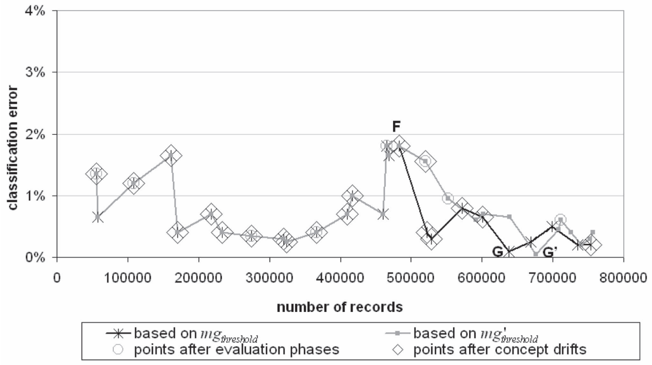

Fig. 16. Forest classification errors for PhotoObj.

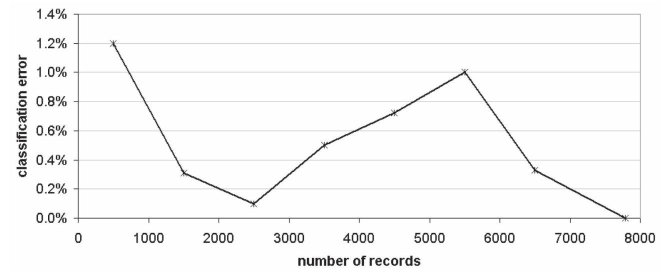

Fig. 17. Classification errors of the Random Forests for PhotoObj.

Although the algorithm detects many concept drift points (marked by diamonds), the classification error rates are maintained throughout the run. The error rates, however fluctuate, at a minimum of 0.05 percent at point G for the plot based on $mg_{threshold}$ and 0.1 percent at point G′ for the plot based on $mg'_{threshold}$.

To check that the classification error is reasonable, we select one percent of the records randomly, and divide the resulting data set into eight small sets. Each small data set, which samples a subinterval of the stream, contains 1,000 records, except the final data set, which contains 786 records. We ran the standard Random Forests algorithm on each data set 10 times, and recorded the classification error for each. Fig. 17 displays the average classification errors.

The figure shows a graph with a maximum error of 1.2 percent, and a minimum of 0 percent. It has a similar structure of Fig. 16; the peaks and valleys of both figures appear at about the same points in the sequence. This shows that the classification accuracy of the streaming algorithm follows a similar trend to the classification accuracy of the standard algorithm.

To better show the algorithm's behavior of detecting concept changes for real data sets without losing classification accuracy, we conduct an experiment using the SpecPhoto data set. The values of $\overline{mg}$ for this experiment exceed $mg_{threshold}$ and $mg'_{threshold}$, each time the forest is considered for evaluation. We, therefore, only consider one graph for the two runs.

Fig. 18 shows the normalized entropy H', the entropy accumulated average H_{AVG}, and the entropy threshold $H_{AVG} + \gamma$, for $\gamma = 0.25$. The size of the entropy windows for this experiment is 500.

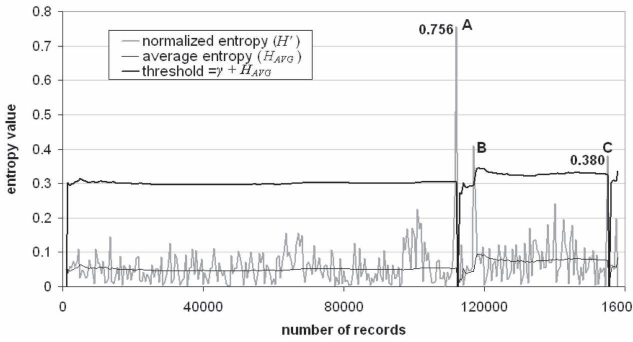

Fig. 18. Entropy values for SpecPhoto.

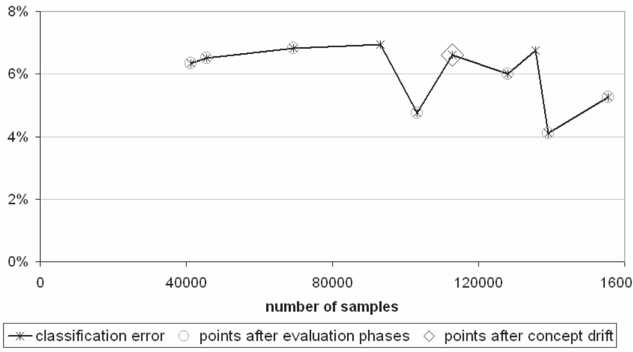

Fig. 19. Forest classification errors for SpecPhoto.

The figure shows that the algorithm detects a concept drift with a significant entropy value of 0.756 at record 112,000 (point A), which represents the concept drift of the data. Recall that the point of concept drift appears at record 111,353. The figure also shows two other points of concept drift (points B and C). The algorithm does not, however, consider point B as a concept drift point since it appears very close to the drift at point A. We believe that the drift appears at point C as a result of the rapidly increasing dust when capturing the images of the objects close to the plane of the Milky Way.

Fig. 19 demonstrates the classification errors for the SpecPhoto data set, which records a maximum error of 6.95 percent, and a minimum error of 4.10 percent. Other research work in classification algorithms have used extracted data sets from the SpecPhoto view of the SDSS [34], and recorded classification accuracies for subsets that include only galaxies and stars of about 95 percent (classification error of about five percent), for standard classification algorithms. Our incremental algorithm records an average classification error of six percent. This demonstrates that the algorithm is able to deal with real data sets having concept drifts without losing classification accuracy.

7.7 Time for Building and Updating the Forest

The average total times for the two extensions of the algorithm to process the synthetic data stream are 32.82 and 32.2 minutes. Clearly, both recorded times are in the same range. This demonstrates that the evaluation phases do not add any significant execution time.

8 Conclusions

In this work, we proposed a stream classification ensemble algorithm that is designed to handle Scenario 3 of Fig. 1. The algorithm successfully handles concept changes using an entropy-based concept drift detection technique. It quickly records the new expected classification accuracy after the changes are presented in the stream. It also dynamically adjusts its parameter based on the data seen so far.

The key feature of our algorithm is that it can decide if the forest under construction is robust enough for deployment when a block of labeled records is not long enough to completely update the current forest. A further phase (*evaluation phase*) is defined to evaluate the forest based on classification accuracy, which is calculated using a subset of labeled records (evaluation set), collected during the forest building.

The algorithm then decides if the forest is ready for deployment or not, based on whether the average value of the margin function [29] is greater than a defined threshold. We formulated two definitions of the threshold and tested the algorithm using on both of them. We also formulated two definitions of algorithm confidence, each associated with a threshold definition. One confidence (and threshold) definition represents a linear relation with strict bounds, whereas the other is a monotonic relation with less-conservative bounds.

Our algorithm is fast and incremental. Assuming the trees are balanced, it completes a build/update phase with a time complexity of $\mathcal{O}(U * tree_{min})$, where U is the current number of trees in the forest, and $tree_{min}$ is the minimum number of records used for building a tree.

Experimental results on both synthetic and real data sets demonstrated that the algorithm is able to make appropriate decisions each time a block of labeled records is not long enough to update the current forest completely.

References

[1] Pandora music station, http://www.pandora.com/, 2010.

[2] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, “An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams,” *Proc. 38th Symp. Interface of Statistics,* May 2006.

[3] P. Vorburger and A. Bernstein, “Entropy-Based Concept Shift Detection,” *Proc. Sixth IEEE Int'l Conf. Data Mining (ICDM),* pp. 1113-1118, Dec. 2006.

[4] W. Li, X. Jin, and X. Ye, “Detecting Change in Data Stream: Using Sampling Technique,” *Proc. Third Int'l Conf. Natural Computation (ICNC),* pp. 130-134, Aug. 2007.

[5] A. Bulut and A. Singh, “A Unified Framework for Monitoring Data Streams in Real Time,” *Proc. 21st Int'l Conf. Data Eng. (ICDE),* pp. 44-55, Apr. 2005.

[6] S. Nassar and J. Sander, “Effective Summarization of Multi-Dimensional Data Streams for Historical Stream Mining,” *Proc. 19th Int'l Conf. Scientific and Statistical Database Management (SSDBM),* pp. 30-39, July 2007.

[7] A. Metwally, D. Agrawal, and A. El-Abbadi, “Efficient Computation of Frequent and Top-k Elements in Data Streams,” *Proc. 10th Int'l Conf. Database Theory (ICDT),* pp. 398-412, Jan. 2005.

[8] X. Wang, H. Liu, and J. Han, “Finding Frequent Items in Data Streams Using Hierarchical Information,” *Proc. IEEE Int'l Conf. Systems, Man and Cybernetics (ISIC),* pp. 431-436, Oct. 2007.

[9] P. Domingos and G. Hulten, “Mining High-Speed Data Streams,” *Proc. Sixth ACM SIGKDD,* pp. 71-80, Aug. 2000.

[10] G. Hulten, L. Spencer, and P. Domingos, “Mining Time-Changing Data Streams,” *Proc. Seventh ACM SIGKDD,* pp. 97-106, Aug. 2001.

[11] H. Wang, W. Fan, S. Philip, and J. Han, “Mining Concept-Drifting Data Streams Using Ensemble Classifiers,” *Proc. Ninth ACM SIGKDD,* pp. 226-235, Aug. 2003.

[12] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, “A Multi-Partition Multi-Chunk Ensemble Technique to Classify Concept-Drifting Data Streams,” *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD '09),* Apr. 2009.

[13] F. Chu and C. Zaniolo, “Fast and Light Boosting for Adaptive Mining of Data Streams,” *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD),* pp. 282-292, 2004.

[14] W. Street and Y. Kim, “A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification,” *Proc. Seventh ACM SIGKDD,* pp. 377-382, Aug. 2001.

[15] Y. Sun, G. Mao, X. Liu, and C. Liu, “Mining Concept Drifts from Data Streams Based on Multi-Classifiers,” *Proc. 21st Int'l Conf. Advanced Information Networking and Applications Workshops (AINAW),* pp. 257-263, May 2007.

[16] Z. Li, T. Wang, R. Wang, Y. Yan, and H. Chen, “A New Fuzzy Decision Tree Classification Method for Mining High-Speed Data Streams Based on Binary Search Trees,” *Proc. First Int'l Frontiers in Algorithmics WorkShop (FAW),* pp. 216-227, Aug. 2007.

[17] C.J. Tsai, C.I. Lee, and W.P. Yang, "An Efficient and Sensitive Decision Tree Approach to Mining Concept-Drifting Data Streams," *Informatica,* vol. 19, no. 1, pp. 135-156, Feb. 2008.

[18] Y. Liu, J. Cai, J. Yin, and A.W.-C. Fu, "Clustering Text Data Streams," *J. Computer Science Technology,* vol. 23, no. 1, pp. 112-128, Jan. 2008.

[19] K. Udommanetanakit, T. Rakthanmanon, and K. Waiyamai, "E-Stream: Evolution-Based Technique for Stream Clustering," *Proc. Third Int'l Conf. Advanced Data Mining and Applications (ADMA),* pp. 605-615, Aug. 2007.

[20] Y. Zhang and X. Jin, "An Automatic Construction and Organization Strategy for Ensemble Learning on Data Streams," *SIGMOD Record,* vol. 35, no. 3, pp. 28-33, Sept. 2006.

[21] F. Chu, Y. Wang, and C. Zaniolo, "An Adaptive Learning Approach for Noisy Data Streams," *Proc. Fourth IEEE Int'l Conf. Data Mining (ICDM),* pp. 351-354, Nov. 2004.

[22] K. Nishida, K. Yamauchi, and T. Omori, "ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments," *Proc. Sixth Int'l Workshop Multiple Classifier Systems (MCS),* pp. 176-185, June 2005.

[23] H. Abdulsalam, D. Skillicorn, and P. Martin, "Streaming Random Forests," *Proc. 11th Int'l Database Eng. and Applications Symp. (IDEAS),* pp. 225-232, Sept. 2007.

[24] H. Abdulsalam, D. Skillicorn, and P. Martin, "Classifying Evolving Data Streams Using Dynamic Streaming Random Forests," *Proc. 19th Int'l Conf. Database and Expert Systems Applications (DEXA),* pp. 643-651, Sept. 2008.

[25] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Wadsworth Int'l, 1984.

[26] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables," *J. Am. Statistical Assoc.,* vol. 58, no. 1, pp. 13-30, 1963.

[27] M. Kantardzic, *Data Mining: Concepts, Models, Methods, and Algorithms.* IEEE Press, 2003.

[28] C.E. Shannon, "A Mathematical Theory of Communication," *Proc. ACM SIGMOBILE,* vol. 5, no. 1, p. 355, Jan. 2001.

[29] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, Oct. 2001.

[30] Random Forest FORTRAN Code, http://www.stat.berkeley.edu/breiman/RandomForests/cc_software.htm/, http://www.stat.berkeley.edu/~breiman/RandomForests/cc_software.htm, 2010.

[31] G. Melli, "(SCDS-A) Synthetic Classification Data Set Generator," Simon Fraser Univ., School of Computer Science, http://www.datasetgenerator.com/, 1997.

[32] The Sloan Digital Sky Survey (SDSS), http://www.sdss.org, 2010.

[33] "The Sixth Data Release of the Sloan Digital Sky Survey," *The Astrophysical J. Supplement Series,* vol. 175, pp. 297-313, Apr. 2008.

[34] S. McConnell and D. Skillicorn, "Distributed Data Mining for Astrophysical Datasets," *Proc. Astronomical Data Analysis Software and Systems XIV,* P. Shopbell, M. Britton, and R. Ebert, eds., vol. 347, pp. 360-364, Dec. 2005.

Hanady Abdulsalam received the BSc and MSc degrees in computer engineering from Kuwait University, and the PhD degree from the School of Computing, Queen's University. She is an assistant professor in the Information Science Department, Kuwait University. Her research interests are in the area of databases and data mining. She also works in the area of scheduling algorithms.

David B. Skillicorn received the PhD degree from the University of Manitoba. He is a professor in the School of Computing at Queen's University, where he heads the Smart Information Management Laboratory. He is also an adjunct professor at the Royal Military College of Canada. His research interests are in data mining, particularly for counterterrorism and fraud; he has also worked extensively in parallel and distributed computing. He is a member of the IEEE.

Patrick Martin received the BSc degree from the University of Toronto, the MSc degree from Queen's University, and the PhD degree from the University of Toronto. He is a professor in the School of Computing at Queen's University. He is the principal investigator on joint research projects with IBM Canada and CA Canada. He is also a visiting scientist with IBM's Centre for Advanced Studies. His research interests include database system performance, Web services and services management, and autonomic computing systems. He is a member of the IEEE Computer Society.