Extremely Fast Decision Tree Mining for Evolving Data Streams

Albert Bifet
LTCI, Télécom ParisTech
Université Paris-Saclay
Paris, France 75013

Jiajin Zhang
HUAWEI Noah's Ark Lab
Hong Kong

Wei Fan
Baidu Research Big Data Lab
Sunnyvale, CA

Cheng He, Jianfeng Zhang
HUAWEI Noah's Ark Lab
Hong Kong

Jianfeng Qian
Columbia University
New York, NY

Geoff Holmes,
Bernhard Pfahringer
University of Waikato
Hamilton, New Zealand

ABSTRACT

Nowadays real-time industrial applications are generating a huge amount of data continuously every day. To process these large data streams, we need fast and efficient methodologies and systems. A useful feature desired for data scientists and analysts is to have easy to visualize and understand machine learning models. Decision trees are preferred in many real-time applications for this reason, and also, because combined in an ensemble, they are one of the most powerful methods in machine learning.

In this paper, we present a new system called streamDM-C++, that implements decision trees for data streams in C++, and that has been used extensively at Huawei. Streaming decision trees adapt to changes on streams, a huge advantage since standard decision trees are built using a snapshot of data, and can not evolve over time. streamDM-C++ is easy to extend, and contains more powerful ensemble methods, and a more efficient and easy to use adaptive decision trees. We compare our new implementation with VFML, the current state of the art implementation in C, and show how our new system outperforms VFML in speed using less resources.

CCS CONCEPTS

• **Information systems → Data stream mining**;

KEYWORDS

Data Streams; Online Learning; Decision Trees; Classification

ACM Reference format:
Albert Bifet, Jiajin Zhang, Wei Fan, Cheng He, Jianfeng Zhang, Jianfeng Qian, and Geoff Holmes, Bernhard Pfahringer. 2017. Extremely Fast Decision Tree Mining for Evolving Data Streams. In *Proceedings of KDD '17, Halifax, NS, Canada, August 13-17, 2017,* 10 pages.
https://doi.org/10.1145/3097983.3098139

KDD '17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098139

1 INTRODUCTION

Streaming data analysis in real time is becoming the fastest and most efficient way to obtain useful knowledge from what is happening now, allowing organizations to react quickly when problems appear or to detect new trends helping to improve their performance. One example is the data produced in mobile broadband networks. A metropolitan city in China can have nearly 10 million subscribers who continuously generate massive data streams in both signalling and data planes into the network. Even with summarisation and compression, the daily volume of the data can reach over 10TB, among which 95% is in the format of sequential data streams. The high volume of data overwhelms the computational resources for processing the data. Even worse, the data streams may experience a change of statistical distribution over time (this phenomenon is known as a concept drift), because of the dynamics of subscriber behaviours and network environments. While it is critical to analyze the data in order to obtain better insights into network management and planning, how to effectively mine the knowledge from the large-scale data streams is a non-trivial issue.

In the data stream model, we need to deal with resources in an efficient and low-cost way. We are interested in three main dimensions:

- accuracy
- amount of space (computer memory) necessary
- the time required to learn from training examples and to predict

These dimensions are typically interdependent: adjusting the time and space used by an algorithm can influence accuracy. By storing more pre-computed information, such as look up tables, an algorithm can run faster at the expense of space. An algorithm can also run faster by processing less information, either by stopping early or storing less; the more time an algorithm has, the more likely it is that accuracy can be increased.

Classification is one of the most widely used data mining techniques. In very general terms, given a list of groups (often called classes), classification seeks to predict to which group a new instance may belong. The outcome of classification is typically either the identification of a single group or the production of a probability distribution of likelihood of membership of each group. A spam filter is a good example, where we want to predict if new emails are considered spam or not. Twitter sentiment analysis is another example, where we want to predict if the sentiment of a new incoming tweet is positive or negative. The probability distribution

generating the data may be changing, and this is why streaming methods need to be adaptive to the changes on the streams [23, 24].

More formally, the classification problem can be formulated as follows: given a set of instances of the form (x, y), where $x = x_1, \ldots, x_k$ is a vector of attribute values, and y is a discrete class from a set of n_C different classes, the classifier builds a model $y = f(x)$ to predict the classes y of future examples. For example, x could be a tweet and y the polarity of its sentiment; or x could be an email message, and y the decision whether it is spam or not.

The state-of-the-art methods for classification of evolving data streams are classifiers based on decision trees. A *Hoeffding tree* [8] is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the information gain of an attribute).

The VFML (Very Fast Machine Learning) toolkit was the first open-source framework for mining high-speed data streams and very large data sets. VFML is written mainly in standard C, and contains tools for learning decision trees (VFDT and CVFDT), for learning Bayesian networks, and for clustering.

At Huawei, many real data applications need fast decision trees that use a small amount of memory. C and C++ are still considered the languages preferred for high-performance applications. Using VFML for data stream mining was not good enough in large-scale applications of mobile data, so there was a need to create a more efficient system for Huawei applications. In this paper, we present a new framework streamDM-C++, that implements extremely fast decision trees in C++, that outperforms VFML, and that contains more classification algorithms than VFML.

The main advantages of streamDM-C++ over VFML are the following:

- Evaluation and classifiers are separated, not linked together.
- It contains several methods for learning numeric attributes.
- It is easy to extend and add new methods.
- The adaptive decision tree is more accurate and does not need an expert user to choose optimal parameters to use.
- It contains powerful ensemble methods as Bagging, Boosting, and Random Forests.
- It is much faster and uses less memory.

This paper is structured as follows. We present VFML in Section 2, streamDM-C++ in Section 3. We perform an empirical evaluation in Section 4, and finally, in Section 6 we give our conclusions.

2 VFML: VERY FAST MACHINE LEARNING TOOLKIT

The VFML (Very Fast Machine Learning) toolkit was the first open-source framework for mining high-speed data streams and very large data sets. It was developed before 2004. VFML is made up of three main components:

- a collection of tools and APIs that help a user develop new learning algorithms

```
HoeffdingTree(Stream, δ)
1  ▷ Let HT be a tree with a single leaf(root)
2  ▷ Init counts n_ijk at root
3  for each example (x, y) in Stream
4       do HTGrow((x, y), HT, δ)

HTGrow((x, y), HT, δ)
1  ▷ Sort (x, y) to leaf l using HT
2  ▷ Update counts n_ijk at leaf l
3  if examples seen so far at l are not all of the same class
4     then
5          ▷ Compute G for each attribute
6          if G(Best Attr.)−G(2nd best) > sqrt(R^2 ln 1/δ / 2n)
7            then
8                 ▷ Split leaf on best attribute
9                 for each branch
10                     do ▷ Start new leaf
11                           and initiliatize counts
```

Figure 1: The Hoeffding Tree algorithm

- a collection of implementations of important learning algorithms
- a collection of scalable learning algorithms that were developed by Pedro Domingos and Geoff Hulten [8].

VFML is written mainly in standard C, and provides a series of tutorials and examples as well as in-source documentation in JavaDoc format. VFML contains tools for learning decision trees (VFDT and CVFDT), for learning Bayesian networks, and for clustering. VFML is being distributed under a modified BSD license.

VFML was the first software framework for data streams, but the main motivation for it was to be a test bed for the academic papers of the authors, not open-source software to be extended. For example, evaluation methods are inside classifiers, and there are significant barriers in terms of creating new methods.

2.1 The Hoeffding Tree

In the data stream setting, where we can not store all the data, the main problem of building a decision tree is the need of reusing the examples to compute the best splitting attributes. Hulten and Domingos [8] proposed the Hoeffding Tree or VFDT, a very fast decision tree for streaming data, where instead of reusing instances, we wait for new instances to arrive. The most interesting feature of the Hoeffding tree is that it has theoretical guarantees that it can build an identical tree with a traditional one if the number of instances is large enough.

The pseudo-code of VFDT is shown in Figure 1. The Hoeffding Tree is based on the Hoeffding bound. This inequality or bound justifies that a small sample can often be enough to choose an optimal splitting attribute. Suppose we make n independent observations of a random variable r with range R, where r is an attribute selection measure such as information gain or Gini impurity gain. The Hoeffding inequality states that with probability $1 - \delta$, the true mean $\bar{r}$ of r will be at least $E[r] - \epsilon$, with

$$\epsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$$

Using this fact, the Hoeffding tree algorithm can determine, with high probability, the smallest number n of examples needed at a node when selecting a splitting attribute.

The Hoeffding Tree maintains in each node the statistics needed for splitting attributes. For discrete attributes, this is the same information as needed for computing the Naïve Bayes predictions: a 3-dimensional table that stores for each triple (x_i, v_j, c) a count $n_{i,j,c}$ of training instances with $x_i = v_j$, together with a 1-dimensional table for the counts for each class. The memory needed depends on the number of leaves of the tree, but not on the size of the data stream.

A theoretically appealing feature of Hoeffding Trees not shared by other incremental decision tree learners is that it has sound guarantees of performance. Using the Hoeffding bound one can show that its output is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples.

Domingos et al. [8] improved the Hoeffding Tree algorithm with an extended method called VFDT, with the following characteristics:

- Ties: when two attributes have similar split gain G, the improved method splits if the Hoeffding bound computed is lower than a certain threshold parameter τ.

$$G(\text{Best Attr.}) - G(\text{2nd best}) < \sqrt{\frac{R^2 \ln 1/\delta}{2n}} < \tau$$

- To speed up the process, instead of computing the best attributes to split every time a new instance arrives, it computes them every time a number n_{min} of instances has arrived.
- To reduce the memory used in the mining, it deactivates the least promising nodes that have lower $p_l \times e_l$ where
  - p_l is the probability to reach leaf l
  - e_l is the error in the node l
  - It is possible to initialize the method with an appropriate decision tree. Hoeffding Trees can grow slowly and performance can be poor initially so this extension provides an immediate boost to the learning curve

A way to improve the classification performance of the Hoeffding Tree is to use Naïve Bayes learners at the leaves instead of the majority class classifier. Gama et al. [10] were the first to use Naïve Bayes in Hoeffding Tree leaves, replacing the majority class classifier. However, Holmes et al. [15] identified situations where the Naïve Bayes method outperformed the standard Hoeffding tree initially but is eventually overtaken. To solve that, they proposed a hybrid adaptive method that generally outperforms the two original prediction methods for both simple and complex concepts.

This method works by performing a Naïve Bayes prediction per training instance, and comparing its prediction with the majority class. Counts are stored to measure how many times the Naïve 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 Naïve Bayes prediction if it has been more accurate overall than the majority class, otherwise it resorts to a majority class prediction.

2.2 Concept-adapting Very Fast Decision Trees CVFDT

Hulten, Spencer and Domingos presented the CVFDT (Concept-adapting Very Fast Decision Trees) algorithm [16] as an extension of VFDT to deal with concept drift, maintaining a model that is consistent with the instances stored in a sliding window. It does not have theoretical guarantees like the Hoeffding Tree.

Figure 2 shows the code for the CVFDT algorithm. It is similar to the code for the Hoeffding Tree but with the following changes:

- The main method maintains a sliding window with the latest instances to have arrived: so it has to add, remove and forget instances.
- The main method in addition to CVFDTGrow, calls also another method called CheckSplitValidity to check if the splits chosen are still valid.
- CVFDTGrow also updates counts of the nodes traversed in the sort.
- CheckSplitValidity creates an alternate subtree if the attributes chosen to split are now different from the ones that were chosen when the split was done.
- Periodically, it checks if the alternate branch is performing better than the original branch tree, and if it is performing better it replaces it, and if not, it removes the alternate branch.

```
CVFDT(Stream, δ)
1  ▷ Let HT be a tree with a single leaf(root)
2  ▷ Init counts n_ijk at root
3  for each example (x, y) in Stream
4       do Add, Remove and Forget Examples
5          CVFDTGrow((x, y), HT, δ)
6          CheckSplitValidity(HT, n, δ)

CVFDTGrow((x, y), HT, δ)
1  ▷ Sort (x, y) to leaf l using HT
2  ▷ Update counts n_ijk at leaf l and nodes traversed in
3    the sort
4  if examples seen so far at l are not all of the same class
5    then
6         ▷ Compute G for each attribute
7         if G(Best Attr.)−G(2nd best) > sqrt(R^2 ln 1/δ / 2n)
8           then
9                ▷ Split leaf on best attribute
10               for each branch
11                    do ▷ Start new leaf and
12                         initialize counts
13               ▷ Create alternate subtree

CheckSplitValidity(HT, n, δ)
1  for each node l in HT that it is not a leaf
2       do for each tree T_alt in ALT(l)
3             do CheckSplitValidity(T_alt, n, δ)
4          if exists a new promising attributes at node l
5             do ▷ Start an alternate subtree
```

Figure 2: The CVFDT algorithm

3 STREAMDM-C++ DATA STREAM MINING SYSTEM

The main goals of streamDM-C++ is to build a fast and efficient system, well designed to be very easy to use and to extend. The main objectives of streamDM-C++ are the following:

- run experiments from the command line like this:
  `PrequentialEvaluation -t train.dataset -e test.dataset -l HoeffdingTree -s (FileReader -f file)`
- add new learners easily
- create new tasks easily
- evaluators and learners should be separated, not linked together
- contain several methods for learning numeric attributes
- the adaptive decision tree should not need to tune parameters that depend on the dataset

Instead of using CVFDT, streamDM-C++ uses the Hoeffding Adaptive Tree [3], since it adapts to the scale of time change in the data, and it does not need an experienced user to select the correct parameters. In the rest of this section, we describe the advantages of the Hoeffding Adaptive Tree, and explain in detail the design of streamDM-C++.

3.1 Hoeffding Adaptive Tree

The *Hoeffding Adaptive Tree* [3] is an adaptive extension to the Hoeffding Tree, that has theoretical guarantees, no need of parameters, and uses ADWIN as a change detector and error estimator. A formal and quantitative statement of the theoretical guarantees (in the form of a theorem) of the Hoeffding Adaptive Tree appears in [3].

ADWIN [2] 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".

More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: first, that change is reliably declared whenever the window shrinks; and second, that at any time the average over the existing window can be reliably taken as an estimate of the current average in the stream (barring a very small or very recent change that is still not statistically visible).

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, a property inherent to all algorithms dealing with random processes.

Also, ADWIN does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique. This means that it keeps a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item.

CVFDT has no theoretical guarantees, and it uses a number of parameters, with default values that can be changed by the user - but which are fixed for a given execution. Besides the example window length, it needs:

(1) T_0: after each T_0 examples, CVFDT traverses the entire decision tree, and checks at each node if the splitting attribute is still the best. If there is a better splitting attribute, it starts growing an alternate tree rooted at this node, and it splits on the currently best attribute according to the statistics at the node.
(2) T_1: after an alternate tree is created, the following T_1 examples are used to build the alternate tree.
(3) T_2: after the arrival of T_1 examples, the following T_2 examples are used to test the accuracy of the alternate tree. If the alternate tree is more accurate than the current one, CVDFT replaces it with this alternate tree (we say that the alternate tree is promoted).

The default values are $T_0 = 10,000$, $T_1 = 9,000$, and $T_2 = 1,000$. One can interpret these figures as the presumption that often the last $50,000$ examples are likely to be relevant, and that change is not likely to occur faster than every $10,000$ examples. These presumptions may or may not be correct for a given data source.

The main internal differences of the Hoeffding Adaptive Tree with respect to CVFDT are:

- The alternate trees are created as soon as change is detected, without having to wait for a fixed number of examples to arrive after the change. Furthermore, the more abrupt the change, the faster a new alternate tree will be created.
- Hoeffding Adaptive Tree replaces the old trees by the new alternate trees as soon as there is evidence that they are more accurate, rather than having to wait for another fixed number of examples.

These two effects can be summarized by saying that the Hoeffding Adaptive Tree adapts to the scale of time change in the data, rather than having to rely on the *a priori* guesses made by the user.

In the case of noisy distributions with outliers, the Hoeffding Adaptive Tree and CVFDT will not fluctuate abruptly when they see an outlier, since they compute and use mean values that helps to smooth the computation.

3.2 Architecture

The architecture of streamDM-C++ is composed by the following elements:

Instances Data arrive as a sequence of instances. Instances contain values for a set of input attributes, and for only one output attribute. streamDM-C++ contains two types of instances
- Dense Instance: stores the information of the instance using an array of doubles..
- Sparse Instance: only stores the non-zero values of an instance using two arrays, one for the indices, and one for the values. An InstanceHeader object maintains meta-information about attributes using Attribute objects.

Streams Streams are the objects that read and get instances. Instances can be obtained in ARFF, C4.5, or CSV format.

Learner Every algorithm in streamDM-C++ is a learner, that basically has three methods:
- init(): to reset and initialize the model when there is change or at the beginning of the mining process

- update(instance): to update the model when a new instance arrives
- predict(instance): to predict the probabilities for each class label

Creating new classifiers is as simple as implementing these three methods.

Evaluator Evaluators maintain and keep statistics of the measurements of the results. We have two types:

- BasicClassificationPerformanceEvaluator: keeps statistics over a landmark window, i.e., from the beginning of the data mining rocess.
- WindowClassificationPerformanceEvaluator: keeps statistics over a sliding window containing the most recent predictions and results.

Task Tasks contains workflows of the stream mining processes. streamDM-C++ runs tasks. As example, for evaluation we have two important tasks:

- Holdout Evaluation: performs an evaluation using different test and training datasets
- Prequential Evaluation: performs an interleaved test then train evaluation. All instances are first used to test and then to train

Parameters To be able to execute tasks, on the command line, we need to be able to use classes as parameters. An important feature is to implement class parameters recursively, so that classes can have class parameters as well. For example, a bagging class can have the decision tree class as its base learner and a Naive Bayes Classifier class at the leaves.

3.3 Machine Learning Algorithms

In this section we detail the learners available in streamDM-C++ and the classes needed to implement the decision trees.

The classifiers available in streamDM-C++ are the following:

HoeffdingTree This is the main class that manages the initialization, the growing and the prediction of the Hoeffding tree. It is a learner with a tree structure of nodes.

HoeffdingAdaptiveTree This is the main class for the adaptive decision tree, that extends the Hoeffding tree.

Bagging Ensemble that uses sampling with replacement, and majority vote for prediction.

Boosting Streaming boosting that approximates the batch traditional boosting classifier.

Random Forests Bagging that uses randomized decision trees.

The components of the Hoefffding Tree are the following:

AttributeClassObserver Statistics for each attribute that manages information about the distribution of the data for each class label. We give more details in the next subsection.

SplitCriterion Criterion to decide splits. There are two different criteria: information gain, and Gini, but it is very easy to add new ones.

InstanceConditionalTest Tests used to mark the branches.

Node Node of the tree, that can be a leaf (active learning node), or an internal node (split node).

ActiveLearningNode Node at a leaf of the tree, that keeps statistics using AttributeClassObservers needed to decide if we need to split or not.

SplitNode Node that is no longer a leaf, that contains branches with an InstanceConditionalTest for each branch.

3.4 Handling numeric attributes

Handling numeric attributes in a data stream classifier, is much more difficult than in a non-streaming setting. In this section we will present the most popular methods used in decision tree algorithms in evolving data streams. They are all implemented in streamDM-C++, as AttributeClassObserver objects. We look at how to manage the statistics of numeric attributes, and how to decide what are the best splitting points in decision trees.

3.4.1 VFML. VFML contains the following method for handling numeric attributes in VFDT and CVFDT: basically, numeric attribute values are summarized by a set of ordered bins. The range of values covered by each bin is fixed at creation and does not change as more examples are seen. A hidden parameter serves as a limit on the total number of bins allowed–in the VFML implementation this is hard-coded to allow a maximum of one thousand bins. Initially, for every new unique numeric value seen, a new bin is created. Once the fixed number of bins have been allocated, each subsequent value in the stream updates the counter of the nearest bin.

There are two potential issues with the approach. Clearly, the method is sensitive to data order. If the first one thousand examples seen in a stream happen to be skewed to one side of the total range of values, then the final summary will be incapable of accurately representing the full range of values.

The other issue is estimating the optimal number of bins. Too few bins will mean the summary is small but inaccurate, whereas too many bins will increase accuracy at the cost of space. In the experimental comparison the maximum number of bins is varied to test this effect.

3.4.2 Exhaustive Binary Tree. This method represents the case of achieving perfect accuracy at the necessary expense of storage space. The decisions made are the same that a batch method would make, because essentially it is a batch method—no information is discarded other than the observed order of values.

Gama et al. present this method in their *VFDTc* system [9]. It works by incrementally constructing a binary search tree structure as values are observed. The path a value follows down the tree depends on whether it is less than, equal to or greater than the value at a particular node in the tree. The values are implicitly sorted as the tree is constructed.

This structure saves space over remembering every value observed at a leaf when a value that has already been recorded reappears in the stream. In most cases a new node will be introduced to the tree. If a value is repeated the counter in the binary search tree node responsible for tracking that value can be incremented. Even then, the overhead of the tree structure will mean that space can only be saved if there are many repeated values. If the number of unique values were limited, as is the case in some data sets, then the storage requirements will be less intensive.

The primary function of the tree structure is to save time. It lowers the computational cost of remembering every value seen, but does little to reduce the space complexity. The computational

considerations are important, because a slow learner can be even less desirable than one that consumes a large amount of memory.

Beside memory cost, this method has other potential issues. Because every value is remembered, every possible threshold is also tested when the information gain of split points is evaluated. This makes the evaluation process more costly than more approximate methods.

This method is also prone to data order issues. The layout of the tree is established as the values arrive, such that the value at the root of the tree will be the first value seen. There is no attempt to balance the tree, so data order is able to affect the efficiency of the tree. In the worst case, an ordered sequence of values will cause the binary search tree algorithm to construct a list, which will lose all the computational benefits compared to a well balanced binary search tree.

3.4.3 Greenwald and Khanna Quantile Summaries. The field of database research is also concerned with the problem of summarizing the numeric distribution of a large data set in a single pass and limited space. The ability to do so can help to optimize queries over massive databases.

Greenwald and Khanna [13] proposed a *quantile summary* method with even stronger accuracy guarantees than previous approaches. The method works by maintaining an ordered set of tuples, each of which records a value from the input stream, along with implicit bounds for the range of each value's true rank. Specifically, a tuple $t_i = (V_i, g_i, \Delta_i)$ consists of three values:

- a value v_i of one of the elements seen so far in the stream
- a value g_i that equals $r_{min}(v_i) - r_{min}(v_{i-1})$, where $r_{min}(v)$ is the lower bound of the rank of v among all the values seen so far
- a value Δ_i that equals $r_{max}(v_i) - r_{min}(v_i)$, where $r_{max}(v)$ is the upper bound of the rank of v among all the values seen so far

Note that $r_{min}(v_i) = \sum_{j \leq i} g_j$, and $r_{max}(v_i) = r_{min}(v_i) + \Delta_i = \sum_{j \leq i} g_j + \Delta_i$.

The quantile summary is said to be ϵ-approximate, after seeing N elements of a sequence any quantile estimate returned will not differ from the exact value by more than ϵN. An operation for compressing the quantile summary is defined, guaranteeing that $\max(g_i + \Delta_i) \leq 2\epsilon N$, so that the error of the summary is kept within a desired bound.

The worst-case space requirement is shown by the authors to be $O(\frac{1}{\epsilon}\log(\epsilon N))$, with empirical evidence showing it to be even better than this in practice.

3.4.4 Gaussian Approximation. This method presented in [20] approximates a numeric distribution in small constant space, using a *Gaussian* (commonly known as *normal*) distribution. Such a distribution can be incrementally maintained by storing only two numbers in memory, and is completely insensitive to data order. A Gaussian distribution is essentially defined by its mean value, which is the center of the distribution, and standard deviation or variance, which is the spread of the distribution. The shape of the distribution is a classic bell-shaped curve that is known by scientists and statisticians to be a good representation of certain types of natural phenomena, such as the weight distribution of a population of organisms.

For each numeric attribute the numeric approximation procedure maintains a separate Gaussian distribution per class label. A method similar to this is described by Gama et al. in their UFFT system [10]. To handle more than two classes, the system builds a forest of trees, one tree for each possible pair of classes. When evaluating split points in that case, a single optimal point is computed as derived from the crossover point of two distributions. It is possible to extend the approach, however, to search for split points, allowing any number of classes to be handled by a single tree. The possible values are reduced to a set of points spread equally across the range, between the minimum and maximum values observed. The number of evaluation points is determined by a parameter, so the search for split points is *parametric*, even though the underlying Gaussian approximations are not. For each candidate point the weight of values to either side of the split can be approximated for each class, using their respective Gaussian curves, and the information gain is computed from these weights.

4 COMPARATIVE EXPERIMENTAL EVALUATION

Comparing streaming decision trees with batch trees can be found in the original paper of VFDT, and may not be fair as streaming decision trees may be faster and more accurate when there are changes on the data stream. The settings are different, and also the online evaluation is different from the batch evaluation. On streaming decision trees, only two implementations are available: VFML and MOA.

Massive **O**nline **A**nalysis (MOA) [4] is a software environment for implementing algorithms and running experiments for online learning from data streams in JAVA. JAVA is known to be slower than C++, due to the fact that it needs to run an instance of the Java Virtual Machine. We perform experiments to compare our new framework with these two implementations.

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 data streams of $1,000,000$ instances created using the data generators most commonly found in the literature.

SEA Concepts Generator This artificial dataset contains abrupt concept drift, first introduced in [21]. It is generated using three attributes, where only the two first attributes are relevant. All three 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:

$$(((SEA_9 \oplus_{t_0}^{W} SEA_8) \oplus_{2t_0}^{W} SEA_7) \oplus_{3t_0}^{W} SEA_{9.5})$$

Rotating Hyperplane It was used as testbed for CVFDT versus VFDT in [16]. A hyperplane in d-dimensional space is

Table 1: Time comparison of streamDM-C++ algorithms with the relative percentage of acceleration. Time is measured in seconds. The best individual times are indicated in boldface.

	streamDM-C++				VFML				MOA			
	HT	HAT	HT-NB	HAT-NB	VFDT	CVFDT win 1000	CVFDT win 5000	CVFDT win 10000	HT	HAT	HT-NB	HAT-NB
RBF(50,0)	7.11	18.09	**7.06**	18.16	107.44	23.62	58.49	212.14	14.12	36.68	14.70	37.66
RBF(50,0.0001)	7.24	16.78	**7.02**	17.6	65.99	20.65	50.13	119.78	14.86	34.27	14.54	36.06
RBF(50,0.001)	**7.18**	15.92	7.58	15.65	84.64	20.56	58.17	106.48	15.45	34.09	15.50	33.05
RBF(10,0)	7.14	17.98	**7.06**	17.99	107.74	23.87	58.55	212.69	14.78	38.09	14.62	35.30
RBF(10,0.0001)	**7.10**	17.19	7.28	17.25	92.04	24.39	35.44	124.52	14.46	36.04	14.46	37.11
RBF(10,0.001)	**7.03**	17.53	7.04	17.02	97.01	21.61	35.22	113.23	13.82	37.62	14.69	33.86
SEA(50)	3.76	10.28	**3.64**	10.13	109.13	10.96	41.29	77.91	7.37	21.48	7.41	20.31
SEA(50000)	3.69	10.2	**3.67**	9.97	125.16	10.79	40.86	78.87	7.33	21.56	7.52	19.79
HYP(10,0.001)	6.65	14.95	**6.62**	15.27	51.82	29.46	67.80	111.07	13.65	29.51	14.09	32.70
HYP(10,0.0001)	6.79	12.56	**6.78**	12.8	54.06	19.75	48.25	93.66	13.73	26.01	14.42	27.20
LED(50000)	**6.87**	23.82	6.97	25.25	33.44	12.62	17.00	19.62	14.67	47.49	14.00	53.15
CovType	**0.34**	0.64	0.44	0.73	0.65	1.00	1.55	1.66	0.68	1.31	0.86	1.53
Electricity	7.75	21.56	**7.73**	18.95	22.50	20.05	20.93	26.74	15.53	44.43	16.18	38.71
Poker	**3.99**	7.97	4.01	8.24	21.26	20.95	16.87	18.88	8.23	16.25	8.15	17.37

the set of points x that satisfy

$$\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i$$

where x_i, is the ith coordinate of x. 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. We introduce change to this dataset adding drift to each weight attribute $w_i = w_i + d\sigma$, where σ is the probability that the direction of change is reversed and d is the change applied to every example.

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. The length of the displacement is randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. This effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities. Only numeric attributes are generated. Drift is introduced by moving the centroids with constant speed. This speed is initialized by a drift parameter.

LED Generator This data source originates from the CART book [6]. An implementation in C was donated to the UCI [1] 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. It has an optimal Bayes classification rate of 74%. The particular configuration of the generator used for experiments (led) produces 24 binary attributes, 17 of which are irrelevant.

4.2 Real-World Data

For confidentiality and reproducibility reasons, we use public datasets in our experiments instead of the real ones used at Huawei. The UCI machine learning repository [1] contains some real-world benchmark data for evaluating machine learning techniques. We will consider three: Forest Covertype, Poker-Hand, and Electricity.

Forest Covertype dataset It 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 [12].

Poker-Hand dataset It 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. The order of cards is important, which is why there are 480 possible Royal Flush hands instead of 4.

Electricity dataset Another widely used dataset is the Electricity Market Dataset described by M. Harries [14] and used by Gama [11]. This data was collected from the Australian New South Wales Electricity Market. In this market, the prices are not fixed and are affected by demand and supply of the market. The prices in this market are set every five minutes. The ELEC2 dataset contains 45, 312 instances. The class label identifies the change of the price related to a moving average of the last 24 hours. The class level only reflects deviations of the price on a one day average and removes the impact of longer term price trends.

The size of these datasets is small, compared to tens of millions of training examples of synthetic datasets: 45,312 for ELEC2 dataset, 581,012 for CoverType, and 1,000,000 for Poker-Hand. Another important fact is that we do not know when drift occurs or if there is any drift.

4.3 Results

We use the datasets explained in Sections 4.1 and 4.2. The experiments were performed on 2.66 GHz Core 2 Duo E6750 machines with 4 GB of memory. The evaluation methodology used was Interleaved Test-Then-Train: every example was used for testing the model before using it to train. This interleaved test followed by train procedure was carried out on 1 million examples from the hyperplane, SEA, LED and RandomRBF datasets. 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 first, and baseline, algorithm (HT) is a single Hoeffding tree, enhanced with adaptive Naive Bayes leaf predictions. Parameter settings are $n_{min} = 1000$, $\delta = 10^{-8}$ and $\tau = 0.05$, used in [8]. We compare the following classifiers:

streamDM-C++: Hoeffding Tree (HT), Hoeffding Adaptive Tree (HAT), Hoeffding Tree with Hybrid Adaptive Naive Bayes (HT-NB), and Hoeffding Adaptive Tree with Hybrid Adaptive Naive Bayes (HAT-NB).

VFML: VFDT and CVFDT with different sliding window sizes

MOA: Hoeffding Tree (HT), Hoeffding Adaptive Tree (HAT), Hoeffding Tree with Hybrid Adaptive Naive Bayes (HT-NB), and Hoeffding Adaptive Tree with Hybrid Adaptive Naive Bayes (HAT-NB).

4.3.1 Time. Table 1 reports the speed of the classification models induced on synthetic data and real datasets: Electricity, Forest CoverType, and Poker Hand.

We observe that the speed of CVFDT depends on the size of the window. In general, as the window size increases the time needed also increases. If we compare a single Hoeffding Tree, we see that the streamDM-C++ tree is much faster than the VFML and MOA trees, in some cases 10 times faster than VFML and 2 times faster than MOA. The Hoeffding Adaptive Tree is much slower than the Hoeffding Tree, as it needs more time to manage, detect changes and build new branches.

Comparing the Hoeffding Adaptive Tree with the CVFDT tree, we see that the Hoeffding Adaptive Tree is faster, but the increment of speed is not uniform and depends on the size of the sliding window. Using the hybrid adaptive Naive Bayes at the leaves of both trees, the speed of the Hoeffding Tree and the Hoeffding Adaptive Tree only increases by a small amount.

4.3.2 Memory. The behaviour of the memory results is similar to the behaviour of the time results. Tables 2 and 3 report the memory of the classification models induced on synthetic data and real datasets.

Table 2: Memory comparison of VFML algorithms: VFDT and CVFDT with different sliding window sizes. Memory is measured in Kb.

	VFDT	CVFDT win 1000	CVFDT win 5000	CVFDT win 10000
RBF(50,0)	862,320	**3,040**	13,316	94,920
RBF(50,0.0001)	839,172	**3,032**	15,200	77,028
RBF(50,0.001)	889,348	**3,048**	16,736	35,156
RBF(10,0)	862,320	**3,040**	13,316	94,920
RBF(10,0.0001)	835,464	**3,280**	11,768	85,612
RBF(10,0.001)	866,056	**3,044**	9,816	78,088
SEA(50)	534,616	**1,952**	12,352	31,396
SEA(50000)	545,788	**1,952**	12,052	31,436
HYP(10,0.001)	833,960	**3,052**	17,664	37,852
HYP(10,0.0001)	828,688	**3,056**	17,028	35,760
LED(50000)	1,144,436	**2,192**	7,128	13,196
CovType	36,588	**4,464**	10,776	21,192
Electricity	1,381,460	**3,900**	13,900	27,648
Poker	586,504	**2,404**	5,644	9,404

Table 3: Memory comparison of streamDM-C++ algorithms. Memory is measured in Kb.

	HT	HAT	HT-NB	HAT-NB
RBF(50,0)	**4,828**	6,452	**4,828**	6,328
RBF(50,0.0001)	**4,916**	11,372	**4,916**	11,948
RBF(50,0.001)	**4,512**	9,268	**4,512**	9,084
RBF(10,0)	**4,828**	6,356	**4,828**	6,360
RBF(10,0.0001)	**4,964**	9,184	**4,964**	9,856
RBF(10,0.001)	**4,772**	8,728	**4,772**	9,056
SEA(50)	**4,616**	8,168	**4,616**	8,092
SEA(50000)	**4,600**	8,288	**4,600**	8,236
HYP(10,0.001)	**5,984**	10,364	5,988	9,988
HYP(10,0.0001)	**7,428**	11,776	**7,428**	11,984
LED(50000)	10,284	23,492	**10,280**	24,608
CovType	**3,956**	5,240	3,984	4,932
Electricity	**10,264**	101,124	10,272	76,660
Poker	**5,404**	13,448	5,408	15,316

Comparing a single Hoeffding Tree, we observe that the streamDM-C++ tree needs less memory than the VFML tree.

The memory used by CVFDT depends on the size of the window, as expected. In general, as the window size increases the memory needed also increases. The Hoeffding Adaptive Tree uses more memory than the Hoeffding Tree.

Using the hybrid adaptive Naive Bayes at the leaves of both trees, the memory used by the Hoeffding Tree and the Hoeffding Adaptive Tree only slightly increases.

4.3.3 Accuracy. Table 4 shows the accuracy of decision trees over all the real and synthetic datasets.

In general, we observe the following facts:

- The accuracy of the single Hoeffding trees are similar in VFML and streamDM-C++.
- The performance of the decision trees with hybrid adaptive Naive Bayes at the leaves is superior to the performance of

Table 4: Accuracy comparison of streamDM-C++ algorithms with the relative percentage of acceleration. Accuracy results are given in %. The best individual accuracies are indicated in boldface.

	streamDM-C++				VFML				MOA			
	HT	HAT	HT-NB	HAT-NB	VFDT	CVFDT win 1000	CVFDT win 5000	CVFDT win 10000	HT	HAT	HT-NB	HAT-NB
RBF(50,0)	69.39	74.1	83.18	**84.54**	**69.68**	29.7	37.14	41.71	69.39	74.1	83.18	**84.54**
RBF(50,0.0001)	31.05	35.6	45.27	**61.78**	29.43	29.7	32.06	**33.62**	31.05	35.6	45.27	**61.78**
RBF(50,0.001)	29.88	30.19	32.26	**37.39**	29.71	29.7	**30.70**	29.85	29.88	30.19	32.26	**37.39**
RBF(10,0)	69.39	74.1	83.18	**84.54**	**69.68**	29.7	37.14	41.71	69.39	74.1	83.18	**84.54**
RBF(10,0.0001)	66.72	68.27	**79.23**	79.13	**64.83**	31.32	34.26	42.33	66.72	68.27	**79.23**	79.13
RBF(10,0.001)	65.84	68.22	**76.30**	75.47	**65.68**	29.7	37.17	42.87	65.84	68.22	**76.30**	75.47
SEA(50)	85.63	87.37	86.43	**89.04**	**85.01**	70.41	82.52	83.51	85.63	87.37	86.43	**89.04**
SEA(50000)	85.65	87.33	86.45	**88.70**	**85.01**	70.41	82.52	83.44	85.65	87.33	86.45	**88.70**
HYP(10,0.001)	79.02	79.54	**89.04**	88.92	**75.55**	50.06	63.25	65.25	79.02	79.54	**89.04**	88.92
HYP(10,0.0001)	67.95	76.97	78.76	**87.08**	**60.04**	49.95	55.56	56.36	67.95	76.97	78.76	**87.08**
LED(50000)	43.92	47.62	68.64	**72.60**	**40.04**	15.95	24.52	26.47	43.92	47.62	68.64	**72.60**
CovType	75.72	76.87	79.19	**83.98**	71.63	**76.67**	75.97	75.53	75.72	76.87	79.19	**83.98**
Electricity	67.91	68.26	80.31	**82.29**	64.84	66.24	62.4	**69.83**	67.91	68.26	80.31	**82.29**
Poker	68.09	59.16	**76.06**	66.64	**69.51**	68.97	61.43	58.56	68.09	59.16	**76.06**	66.64

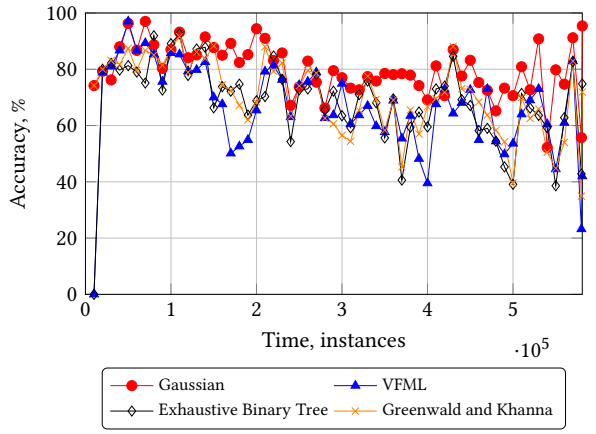

Figure 3: Accuracy comparing different numeric attribute class observers on the Forest Covertype dataset

the decision trees with only the majority class classifier at the leaves.

- The accuracy of CVFDT depend on the size of the window, and the optimal size window is different for each dataset.
- The Hoeffing Adaptive Tree outperforms the HoeffdingTree and CVFDT.

4.4 Numeric Attribute Handlers

streamDM-C++ contains several numeric attribute handlers, in contrast to VFML that contains only one. Figure 3 shows the results of a prequential evaluation using a sliding window of 1,000 examples, on the Forest Covertype dataset, comparing the following attribute class observers: Gaussian approximation, VFML, Exhaustive Binary Tree, and Greenwald and Khanna Quantile Summaries.

For this dataset, we observe that the different numeric attribute handlers perform differently, and that for this specific dataset the Gaussian approximation is the one with highest accuracy. Having several numeric attribute handlers is an important feature of streamDM-C++ that can help to improve accuracy, as it is shown in this example.

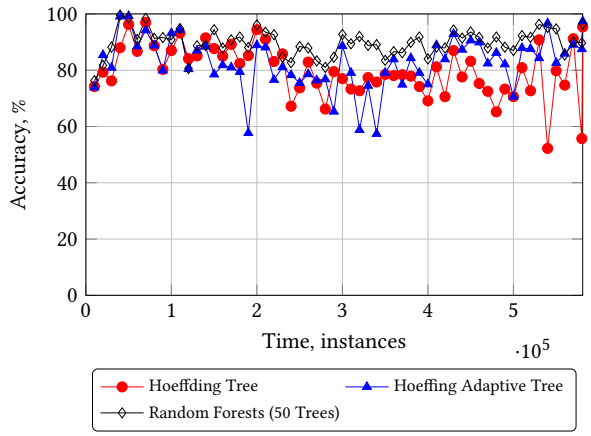

Figure 4: Accuracy comparing the Hoeffding Tree, Hoeffding Adaptive Tree and Random Forests on the Forest Covertype dataset

4.5 Random Forests

Random Forests is a very powerful ensemble method combining a set of decision trees; the Random Forest usually outperforms the single best classifier in the ensemble. Figure 4 shows a comparison of the Hoeffding Tree, Hoeffding Adaptive Tree and Random Forests on the Covertype dataset. We see that the Hoeffding Adaptive Tree has better accuracy than the Hoeffding Tree. More interestingly, the Random Forest classifier outperforms both, confirming that it is an extremely good method for classifying data streams.

5 RELATED WORK

The algorithms in streamDM-C++ are sequential and they are not distributed. There is room for improving the framework using parallelism. Other open source softwares for data stream mining are available that are distributed and work in the Hadoop Ecosystem using Apache Spark, Apache Flink, and Apache Storm.

Apache Scalable Advanced Massive Online Analysis (Apache samoa) [19] is a framework that provides distributed machine learning for big data streams, with an interface to plug-in different stream processing platforms that run in the Hadoop ecosystem.

Apache samoa can be used in two different modes: using it as a running platform to which new algorithms can be added, or developers can implement their own algorithms and run them within their own production system. Another aspect of Apache samoa is the stream processing platform abstraction where developers can also add new platforms by using the available API. With these separation of roles the Apache samoa project is divided into SAMOA API layer and DSPE-adapter layer. The SAMOA API layer allows developers to develop for Apache samoa without worrying about which distributed stream processing engine (SPE) is going to be used. In the case of new SPEs being released or the interest in integrating another platform, a new DSPE-adapter layer module can be added. Apache samoa supports four SPEs that are currently the state or the art: Apache Flink [7], Storm, Samza, and Apex. Apache SAMOA has a parallel VFDT implementation called Vertical Hoeffding Tree [18].

StreamDM for Spark Streaming [5] is an open-source project for mining big data streams using Spark Streaming [22], an extension of the core Spark API that enables scalable stream processing of data streams.

One of the first software available for mining data streams, was the data stream plugin (formerly: concept drift plugin) [17] for RapidMiner (formerly: YALE (Yet Another Learning Environment)), a freely available open-source environment for machine learning, data mining, and knowledge discovery, extends RapidMiner with operators for handling real and simulated concept drift in time-varying data streams.

The data stream mining and concept drift handling operators provided in this plugin can be combined with all other RapidMiner operators. For example, the audio and text preprocessing of the RapidMiner package can be used to detect and handle concept changes in audio and text data streams and all machine learning methods for classification available in RapidMiner (and WEKA) can be combined with the concept drift handling frameworks. However, some of these frameworks require the learners to be able to estimate their classification performance.

6 CONCLUSIONS

In this paper we presented streamDM-C++, a new system for mining evolving streams using decision trees and ensembles in C++. We explained the design choices for the solution, the deployment challenges, and lessons learned. Our experimental validation shows that streamDM-C++ outperforms VFML in the three dimensions of data stream mining processing: time, memory and accuracy.

streamDM-C++ is available as open source software[1], so that practitioners and researchers in industry can benefit of using this new extremely fast implementation in C++ of decision trees for evolving data streams.

REFERENCES

[1] A. Asuncion and D. Newman. UCI machine learning repository, 2007.
[2] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.
[3] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *8th International Symposium on Intelligent Data Analysis*, pages 249–260, 2009.
[4] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis http://moa. cms.waikato.ac.nz/. *Journal of Machine Learning Research (JMLR)*, 2010.
[5] A. Bifet, S. Maniu, J. Qian, G. Tian, C. He, and W. Fan. StreamDM: Advanced data mining in Spark streaming. In *IEEE International Conference on Data Mining Workshop, ICDMW 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 1608–1611, 2015.
[6] L. Breiman et al. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
[7] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.
[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
[9] J. Gama, R. Fernandes, and R. Rocha. Decision trees for mining data streams. *Intell. Data Anal.*, 10(1):23–45, 2006.
[10] J. Gama and P. Medas. Learning decision trees from dynamic data streams. *J. UCS*, 11(8):1353–1366, 2005.
[11] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *SBIA Brazilian Symposium on Artificial Intelligence*, pages 286–295, 2004.
[12] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *KDD '03*, pages 523–528, August 2003.
[13] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM Special Interest Group on Management Of Data Conference*, pages 58–66, 2001.
[14] M. Harries. Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales, 1999.
[15] G. Holmes, R. Kirkby, and B. Pfahringer. Stress-testing hoeffding trees. In *PKDD*, pages 495–502, 2005.
[16] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *7th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.
[17] R. Klinkenberg. RapidMiner Data Stream Plugin, 2010.
[18] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo. VHT: Vertical Hoeffding tree. In *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pages 915–922, 2016.
[19] G. D. F. Morales and A. Bifet. SAMOA: Scalable Advanced Massive Online Analysis. *Journal of Machine Learning Research*, 16:149–153, 2015.
[20] B. Pfahringer, G. Holmes, and R. Kirkby. Handling numeric attributes in hoeffding trees. In *Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings*, Lecture Notes in Computer Science, pages 296–307. Springer, 2008.
[21] W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, New York, NY, USA, 2001. ACM Press.
[22] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache Spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.
[23] I. Zliobaite, A. Bifet, M. M. Gaber, B. Gabrys, J. Gama, L. L. Minku, and K. Musial. Next challenges for adaptive learning systems. *SIGKDD Explorations*, 14(1):48–55, 2012.
[24] I. Zliobaite, A. Bifet, J. Read, B. Pfahringer, and G. Holmes. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, 98(3):455–482, 2015.

[1] http://huawei-noah.github.io/streamDM-Cpp/