Journal of Universal Computer Science, vol. 11, no. 8 (2005), 1353-1366
submitted: 5/3/05, accepted: 5/5/05, appeared: 28/8/05 © J.UCS

Learning Decision Trees from Dynamic Data Streams

João Gama
LIACC, FEP - University of Porto
Rua de Ceuta 118-6, 4050 Porto, Portugal
jgama@liacc.up.pt

Pedro Medas
LIACC - University of Porto
Rua de Ceuta 118-6, 4050 Porto, Portugal
pmedas@liacc.up.pt

Abstract: This paper presents a system for induction of forest of functional trees from data streams able to detect concept drift. The Ultra Fast Forest of Trees (UFFT) is an incremental algorithm, which works online, processing each example in constant time, and performing a single scan over the training examples. It uses analytical techniques to choose the splitting criteria, and the information gain to estimate the merit of each possible splitting-test. For multi-class problems the algorithm builds a binary tree for each possible pair of classes, leading to a forest of trees. Decision nodes and leaves contain naive-Bayes classifiers playing different roles during the induction process. Naive-Bayes in leaves are used to classify test examples. Naive-Bayes in inner nodes play two different roles. They can be used as multivariate splitting-tests if chosen by the splitting criteria, and used to detect changes in the class-distribution of the examples that traverse the node. When a change in the class-distribution is detected, all the sub-tree rooted at that node will be pruned. The use of naive-Bayes classifiers at leaves to classify test examples, the use of splitting-tests based on the outcome of naive-Bayes, and the use of naive-Bayes classifiers at decision nodes to detect changes in the distribution of the examples are directly obtained from the sufficient statistics required to compute the splitting criteria, without no additional computations. This aspect is a main advantage in the context of high-speed data streams. This methodology was tested with artificial and real-world data sets. The experimental results show a very good performance in comparison to a batch decision tree learner, and high capacity to detect drift in the distribution of the examples.
Key Words: Data streams, Incremental Decision trees, Concept Drift
Category: H.2.8, I.2.6, I.5.2

1 Introduction

Many sources produce data continuously. Examples include telephone record calls, customer click streams, large sets of web pages, multimedia data, and sets of retail chain transactions. These sources are called data streams. In data streams training examples come over time, usually one at a time. In [7] the authors present desirable properties for learning in data streams: incrementality, online learning, constant time to process each example, single scan over the training set, take drift into account. In these situations is highly unprovable the assumption that the examples are generated at random according to a stationary probability distribution. At least in complex systems and for large time periods, we should expect changes in the distribution of the examples. A natural approach for this *incremental tasks* are *adaptive learning algorithms*, incremental learning algorithms that take into account concept drift.

In this paper we present UFFT, an algorithm that generates forest of functional trees for data streams. The main contributions of this work include a fast method to choose the cut point for splitting tests, use of multivariate splitting tests, the use of functional leaves to classify test cases, and the ability to detect concept drift. These aspects are integrated in the sense that the sufficient statistics needed by the splitting criteria are the only statistics used in the functional leaves, multivariate splitting tests, and in the drift detection method. The paper is organized as follows. In the next section we present related work in the areas of incremental decision-tree induction and concept drift detection. In Section 3, we present the main issues of our algorithm. The system has been implemented, and evaluated in a set of benchmark problems. Preliminary results are presented in Section 4. In the last section we resume the main contributions of this paper, and point out some future work.

2 Related Work

In this section we analyze related work in two dimensions. One dimension is related to methods dealing with concept drift. The other dimension is related to the induction of decision trees from data streams.

In the literature of machine learning, several methods have been presented to deal with time changing concepts [14, 22, 13, 12, 8, 20, 16]. The two basic methods are based on *temporal windows* where the window fixes the training set for the learning algorithm and *weighting examples* that ages the examples, shrinking the importance of the oldest examples. These basic methods can be combined and used together. Both weighting and time window forgetting systems are used for incremental learning. A method to dynamically choose the set of old examples that will be used to learn the new concept faces several difficulties. It has to select enough examples to the learner algorithm and also to keep old data from disturbing the learning process, when older data have a different probability distribution from the new concept. A larger set of examples allows a better generalization if no concept drift happened since the examples arrived [22]. The systems using weighting examples use partial memory to select the more recent examples, and therefore probably within the new context. Repeated examples are assigned more weight. The older examples, according to some threshold, are forgotten and only the newer ones are used to learn the new concept model [12]. When a drift concept occurs the older examples become irrelevant. We can apply a time window on the training examples to learn the new concept description only from the most recent examples. The time window can be improved by adapting its size. Widmer [22] and Klinkenberg [13] present several methods to choose a time window dynamically adjusting the size using heuristics to track the learning process. The methods select the time window to include only examples on the current target concept. Klinkenberg in [12] presents a method to automatically select the time window size in order to minimize the generalization error. Kubat and Widmer [14] describe a system that adapts to drift in continuous domains. Klinkenberg [11] shows the application of several methods of handling concept drift with an adaptive time window on the training data, by selecting representative training examples or by weighting the training examples. Those systems automatically adjust the window size, the example selection and the example weighting to minimize the estimated generalization error.

Concept drift in the context of data streams appears for example in [8, 21, 20]. H. Wang *et al.* train ensembles of batch learners from sequential chunks of data and use error estimates on the test data under the time-evolving environment. G. Hulten and P.Domingos have proposed a method to scale-up learning algorithms to very-large databases [2, 7]. They have presented system VFDT [2], a very fast decision tree algorithm for data-streams described by nominal attributes. The main innovation in VFDT is the use of the Hoeffding bound to decide when a leaf should be expanded to a decision node. The work of VFDT, has been extended with the ability to detect changes in the underlying distribution of the examples. CVFDT [8] is a system for mining decision trees from time-changing data streams. CVFDT works by keeping its model consistent with a sliding window of the most recent examples. When a new example arrives it increments the counts corresponding to the new example and decrements the counts to the oldest example in the window which is now forgotten. Each node in the tree maintains the sufficient statistics. Periodically, the splitting-test is recomputed. If a new test is chosen, the CVFDT starts growing an alternate sub-tree. The old one is replaced only when the new one becomes more accurate.

3 Ultra-Fast Forest Trees - UFFT

UFFT is an algorithm for supervised classification learning that generates a forest of binary trees. The algorithm is incremental, processing each example in constant time, and works online. UFFT is designed for continuous data. It uses analytical techniques to choose the splitting criteria, and the information gain to estimate the merit of each possible splitting-test. For multi-class problems, the algorithm builds a binary tree for each possible pair of classes leading to a forest-of-trees. During the training phase the algorithm maintains a short term memory. Given a data stream, a limited number of the most recent examples are maintained in a data structure that supports constant time insertion and deletion. When a test is installed, a leaf is transformed into a decision node with two descendant leaves. The sufficient statistics of the leaf are initialized with the examples in the short term memory that will fall at that leaf. The UFFT has shown good results with several large and medium size problems. In this work we incorporate in UFFT system the ability to support Concept Drift Detection.

To detect concept drift we maintain, at each inner node, a naive-Bayes classifier [3] trained with the examples that traverse the node. Statistical theory guarantees that for stationary distribution of the examples, the online error of naive-Bayes will decrease; when the distribution function of the examples changes, the online error of the naive-Bayes at the node will increase. In that case we decide that the test installed at this node is not appropriate for the actual distribution of the examples. When this occurs the sub-tree rooted at this node will be pruned. The algorithm forgets the sufficient statistics and learns the new concept with only the examples in the new concept. The drift detection method will always check the stability of the distribution function of the examples at each decision node. In the following sections we provide detailed information about the most relevant aspects of the system.

3.1 Algorithm Details

3.1.1 The Splitting Criteria

The UFFT starts with a single leaf. When a splitting test is installed at a leaf, the leaf becomes a decision node, and two descendant leaves are generated. The splitting test has two possible outcomes each conducting to a different leaf. The value *True* is associated with one branch and the value *False*, with the other. The splitting tests are over a numerical attribute and are of the form $attribute_i \leq value_j$. We use the analytical method for split point selection presented in [15]. We choose, for all numerical attributes, the most promising $value_j$. The only sufficient statistics required are the mean and variance per class of each numerical attribute. This is a major advantage over other approaches, as the exhaustive method used in C4.5 [18] and in VFDTc [6], because all the necessary statistics are computed on the fly. This is a desirable property on the treatment of huge data streams because it guarantees constant time processing each example.

The analytical method uses a modified form of quadratic discriminant analysis to include different variances on the two classes[1]. This analysis assumes that the distribution of the values of an attribute follows a normal distribution for both classes. Let $\phi(\bar{x}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\bar{x})^2}{2\sigma^2}\right)$ be the normal density function,

[1] The reader should note that in UFFT any n-class problem is decomposed into $n(n-1)/2$ two-class problems.

where $\bar{x}$ and σ^2 are the sample mean and variance of the class. The class mean and variance for the normal density function are estimated from the sample set of examples at the node. The quadratic discriminant splits the X-axis into three intervals $(-\infty, d_1)$, (d_1, d_2), (d_2, ∞) where d_1 and d_2 are the possible roots of the equation $p(-)\phi\{(\bar{x}_-, \sigma_-)\} = p(+)\phi\{(\bar{x}_+, \sigma_+)\}$ where $p(i)$ denotes the estimated probability than an example belongs to class i. We pretend a binary split, so we use the root closer to the sample means of both classes. Let d be that root. The splitting test candidate for each numeric attribute i will be of the form $Att_i \leq d_i$. To choose the best splitting test from the candidate list we use an heuristic method. We use the information gain to choose, from all the splitting point candidates, the best splitting test. To compute the information gain we need to construct a contingency table with the distribution per class of the number of examples lesser and greater than d_i:

	$Att_i \leq d_i$	$Att_i > d_i$
Class $+$	p_1^+	p_2^+
Class $-$	p_1^-	p_2^-

The information kept by the tree is not sufficient to compute the exact number of examples for each entry in the contingency table. Doing that would require to maintain information about all the examples at each leaf. With the assumption of normality, we can compute the probability of observing a value less or greater than d_i. From these probabilities and the distribution of examples per class at the leaf we populate the contingency table. The splitting test with the maximum information gain is chosen. This method only requires that we maintain the mean and standard deviation for each class per attribute. Both quantities are easily maintained incrementally. In [6] the authors presented an extension to VFDT to deal with continuous attributes. They use a *Btree* to store continuous attribute-values with complexity $O(nlog(n))$. The complexity of the method proposed here is $O(n)$. This is why we denote our algorithm as *ultra-fast*. Once the merit of each splitting has been evaluated, we have to decide on the expansion of the tree. This problem is discussed in the next section.

3.1.2 From Leaf to Decision Node

To expand the tree, a test $attribute_i \leq d_i$ is installed in a leaf, and the leaf becomes a decision node with two new descendant leaves. To expand a leaf two conditions must be satisfied. The first one requires the information gain of the selected splitting test to be positive. That is, there is a gain in expanding the leaf against not expanding. The second condition, it must exist statistical support in favor of the best splitting test which is asserted using the Hoeffding bound as in VFDT [2]. When new nodes are created, the short term memory is used. This is described in the following section.

3.1.3 Short Term Memory

The short term memory maintains a limited number of the most recent examples. These examples are used to update the statistics at the new leaves when they are created. The examples in the short term memory traverse the tree. Only those that reach the new leaves will update the sufficient statistics of the tree. The data structure used in our algorithm supports constant time insertion of elements at the beginning of the sequence and constant time removal of elements at the end of the sequence.

3.1.4 Functional Leaves

To classify an unlabeled example, the example traverses the tree from the root to a leaf. It follows the path established, at each decision node, by the splitting test at the appropriate attribute-value. The leaf reached classifies the example. The classification method is a *naive-Bayes* classifier. The use of the naive-Bayes classifiers at the tree leaves does not enter any overhead in the training phase. At each leaf we maintain sufficient statistics to compute the information gain. These are the necessary statistics to compute the conditional probabilities of $P(x_i|Class)$ assuming that the attribute values follow, for each class, a normal distribution. Let l be the number of attributes, and $\phi(\bar{x}, \sigma)$ denotes the standard normal density function for the values of attribute i that belong to a given class. Assuming that the attributes are independent given the class, the Bayes rule will classify an example in the class that maximizes the *a posteriori* conditional probability, given by: $P(C^i|\mathbf{x}) \propto \log(Pr(C^i)) + \sum_{k=1}^{l} \log(\phi(\bar{x}_k^i, s_k^i))$. There is a simple motivation for this option. UFFT only changes a leaf to a decision node when there is a sufficient number of examples to support the change. Usually hundreds or even thousands of examples are required. To classify a test example, the majority class strategy only use the information about class distributions and does not look for the attribute-values. It uses only a small part of the available information, a crude approximation to the distribution of the examples. On the other hand, naive-Bayes takes into account not only the prior distribution of the classes, but also the conditional probabilities of the attribute-values given the class. In this way, there is a much better exploitation of the information available at each leaf [6].

3.1.5 Forest of Trees

The splitting criterion only applies to two class problems. Most of real-world problems are multi-class. In the original paper [15] and for a batch-learning scenario, this problem was solved using, at each decision node, a 2-means cluster algorithm to group the classes into two super-classes. Obviously, the cluster method can not be applied in the context of learning from data streams.

We propose another methodology based on round-robin classification [4]. The round-robin classification technique decomposes a multi-class problem into k binary problems, that is, each pair of classes defines a two-class problem. In [4] the author shows the advantages of this method to solve n-class problems. The UFFT algorithm builds a binary tree for each possible pair of classes. For example, in a three class problem (A,B, and C) the algorithm grows a forest of binary trees, one for each pair: A-B, B-C, and A-C. In the general case of n classes, the algorithm grows a forest of $\frac{n(n-1)}{2}$ binary trees. When a new example is received during the tree growing phase each tree will receive the example if the class attached to it is one of the two classes in the tree label. Each example is used to train several trees and neither tree will get all examples. The short term memory is common to all trees in the forest. When a leaf in a particular tree becomes a decision node, only the examples corresponding to this tree are used to initialize the new leaves.

3.1.5.1 Fusion of Classifiers

When doing classification of a test example, the algorithm sends the example to all trees in the forest. The example will traverse the tree from root to leaf and the classification is registered. Each tree in the forest makes a prediction. This prediction takes the form of a probability class distribution. Taking into account the classes that each tree discriminates, these probabilities are aggregated using the *sum rule* [10]. The most probable class is used to classify the example. Note that some examples will be forced to be classified erroneously by some of the binary base classifiers, because each classifier must label all examples as belonging to one of the two classes it was trained on.

3.1.6 Functional Inner-nodes

When evaluating the splitting-criteria the merit of the best attributes could be closed enough that the difference in gains does not satisfy the Hoeffding bound. This aspect has been pointed out in [2]. In VFDT, the authors propose the use of a user defined constant, τ, that can decide towards a split (given that $\epsilon < \tau$), even when the Hoeffding bound is not satisfied. In UFFT when there is a tie in the evaluation of the merit of tests based on single attributes, the system starts trying more complex splitting tests [5].

As we have shown, the sufficient statistics for the splitting-criteria can be directly used to construct a naive-Bayes classifier. The idea of functional inner nodes is to install splitting-tests based on the predictions of the naive-Bayes classifier build at that node.

Suppose that we observe a leaf where the difference in gain between the two best attributes does not satisfies the Hoeffding bound. Since the first tie, when a new training example falls at this leaf, it will be classified using the naive-Bayes derived from the sufficient statistics. Those predictions are used to populate a 2×2 contingency table, where each cell n_{ij} contains the number of examples from class i that naive Bayes predict class j.

In the next evaluation we evaluate also, in addiction to the evaluation of all the original attributes, the information gain of the contingency table obtained by the naive-Bayes predictions. This evaluation corresponds to consider a new attribute: the naive-Bayes predictions. If this implicit attribute is the best attribute in terms of information gain, and the difference with respect to the second best satisfies the Hoeffding bound, then the leaf becomes a decision node with two outcomes: the naive-Bayes predictions. UFFT uses naive-Bayes classifiers at leaves. When considering splitting-tests based on naive-Bayes we must consider the advantage of splitting versus not splitting. For example, if the predictions of naive-Bayes are accurate, the corresponding gain will be high. In such cases, we don't need to expand the leaf, avoiding too much structure and overfitting. After a first tie, we only expand a leaf, if the gain of the naive-Bayes predictions is less than a user defined threshold. In the experiments described below the threshold was set to 0.5. Naive Bayes classifiers use all attributes to make predictions. This aspect could be negative in the presence of irrelevant attributes. In UFFT we only consider splitting-tests based on naive-Bayes classifiers after the first tie. This aspect can be used to select the most informative attributes to use with naive-Bayes.

3.2 Concept Drift Detection

The UFFT algorithm maintains, at each node of all decision trees, a naive-Bayes classifier. Those classifiers were constructed using the sufficient statistics needed to evaluate the splitting criteria when that node was a leaf. When the leaf becomes a node the naive-Bayes classifier will classify the examples that traverse the node. The basic idea of the drift detection method is to control this online error-rate. If the distribution of the examples that traverse a node is stationary, the error rate of naive-Bayes decreases. If there is a change on the distribution of the examples the naive-Bayes error will increase [17]. When the system detect an increase of the naive-Bayes error in a given node, an indication of a change in the distribution of the examples, this suggest that the splitting-test that has been installed at this node is no longer appropriate. In such cases, all the subtree rooted at that node is pruned, and the node becomes a leaf. All the sufficient statistics of the leaf are initialized using the examples in the new context from the short term memory. We designate as *context* a set of contiguous examples where the distribution is stationary, assuming that the data stream is a set of contexts. The goal of the method is to detect when in the sequence of examples of the data stream there is a change from one context to another.

When a new training example becomes available, it will cross the corresponding binary decision trees from the root node till a leaf. At each node, the naive Bayes installed at that node classifies the example. The example will be correctly or incorrectly classified. For a set of examples the error is a random variable from Bernoulli trials. The Binomial distribution gives the general form of the probability for the random variable that represents the number of errors in a sample of n examples. We use the following estimator for the true error of the classification function $p_i \equiv (error_i/i)$ where i is the number of examples and $error_i$ is the number of examples misclassified, both measured in the actual context. The estimate of error has a variance. The standard deviation for a Binomial is given by $s_i \equiv \sqrt{\frac{p_i * (1-p_i)}{i}}$, where i is the number of examples observed within the present context. For sufficient large values of the example size, the Binomial distribution is closely approximated by a Normal distribution with the same mean and variance. Considering that the probability distribution is unchanged when the context is static, then the $1 - \alpha/2$ confidence interval for p with $n > 30$ examples is approximately $p_i \pm \alpha * s_i$. The parameter α depends on the confidence level. In our experiments the confidence level for drift has been set to 99%. The drift detection method manages two registers during the training of the learning algorithm, p_{min} and s_{min}. Every time a new example i is processed those values are updated when $p_i + s_i$ is lower than $p_{min} + s_{min}$.

We use a warning level to define the optimal size of the context window. The context window will contain the old examples that are on the new context and a minimal number of examples on the old context. Suppose that in the sequence of examples that traverse a node, there is an example i with correspondent p_i and s_i. The warning level is reached if $p_i + s_i \geq p_{min} + 1.5 * s_{min}$. The drift level is reached if $p_i + s_i \geq p_{min} + 3 * s_{min}$. Suppose a sequence of examples where the naive-Bayes error increases reaching the warning level at example k_w, and the drift level at example k_d. This is an indicator of a change in the distribution of the examples. A new context is declared starting in example k_w, and the node is pruned becoming a leaf. The sufficient statistics of the leaf are initialized with the examples in the short term memory whose time stamp is greater than k_w. It is possible to observe an increase of the error reaching the warning level, followed by a decrease. We assume that such situations corresponds to a *false alarm*, without changing the context. With this method of learning and forgetting we ensure a way to continuously keep a model better adapted to the present context. The method uses the information already available to the learning algorithm and does not require additional computational resources.

An advantage of this method is it continuously monitors the online error of naive Bayes. It can detect changes in the class-distribution of the examples at any time. All decision nodes contain naive Bayes to detect changes in the class-distribution of the examples that traverse the node, that correspond to detect shifts in different regions of the instance space. Nodes near the root should be able to detect abrupt changes in the distribution of the examples, while deeper nodes should detect smoothed changes.

All the main characteristics of UFFT are due to the splitting criteria. All the statistics required by the splitting criteria can be computed incrementally. Moreover we can directly derive naive Bayes classifiers from the sufficient statistics. Naive Bayes classifiers are used in leaves to classify test examples, are used in inner decision nodes to detect drift and can be used in splitting tests. It is known that naive Bayes is a low-variance classifier. This property is relevant mainly when the naive Bayes acts as splitting test and in the drift detection.

4 Experimental Work

4.1 Stationary Data

The experimental work has been done using the *Waveform*, *LED* and *Balance* datasets available at the UCI repository [1]. There are two *Waveform* problems, both with three classes. The first problem is defined by 21 numerical attributes. The second one contains 40 attributes. It is known that the *optimal Bayes* error is 14%. The *LED* problem has 24 binary attributes (17 are irrelevant) and 10 classes. The *optimal Bayes* error is 26%. The *Balance* problem has 4 attributes and 3 classes. The choice of these datasets was motivated by the existence of dataset generators at the UCI repository that could simulate streams of data. For all the problems we generate training sets of a varying number of examples, starting from 50k till 1500k. The test set contains 100k examples. UFFT generates a model from the training set, seeing each example once. The generated model classifies the test examples. The UFFT algorithm was used with parameters values $\delta = 0.05, \tau = 0.001$, $n_{min} = 300$, and buffer size of 1000. All algorithms ran on a Centrino at 1.5GHz with 512 MB of RAM and using Linux Mandrake.

For comparative purposes, we use C4.5, the *state of the art* in decision tree learning. It is a *batch* algorithm that requires that all the data should fit in memory. All the data is successively re-used at decision nodes in order to choose the splitting test. At each decision node, continuous attributes should be sorted, an operation with complexity of O(n log n). We conducted a set of experiments comparing UFFT against C4.5. Both algorithms learn on the same training dataset, and the generated models are evaluated on the same test set. Detailed results are presented in Table 1. UFFT is orders of magnitude faster that C4.5 generating simpler (in terms of the number of decision nodes) models, with similar performance. The advantage of using multivariate splitting tests is evident in waveform datasets. The differences in performance appears in columns *Version 1* and *Default* in Table 1. In these datasets the observed improvement is about

	Error Rate				**Training Time**		**Tree Size**			
Exs	**UFFT**			**C4.5**	**UFTT**	**C4.5**	**UFFT**			**C4.5**
	No Drift	Version 1	Default							
Balance dataset - 4 Attributes										
100k	3.2	3.3	3.2	3.0	18	41	315	1	1	2929
500k	2.2	2.3	2.2	2.1	128	822	1355	1	1	9689
1000k	1.7	2.1	1.7	1.7	301	2949	2051	1	1	16223
Waveform dataset - 21 Attributes										
100k	15.8	22.4	15.7	19.9	38	156	11	13	1	7945
500k	17.4	23.3	17.3	18.6	223	2734	43	83	65	33787
1000k	17.1	22.9	17.0	18.3	409	10802	113	73	1	61841
Waveform dataset - 40 Attributes										
100k	18.8	25.2	17.4	21.0	69	298	15	7	1	9837
500k	18.4	23.4	18.4	19.5	371	4054	59	53	75	435233
1000k	18.3	23.5	18.3	19.1	719	16346	45	49	1	80331
LED dataset - 24 Attributes										
100k	26.3	26.2	26.2	26.7	174	525	30			8309
500k	26.1	26.1	26.1	26.7	1372	15209	82			34679
1000k	26.1	26.1	26.2	26.7	2861	67271	116			65469

Table 1: Learning curves for the datasets under study. For UFFT we present the results of three versions: disabling drift (No Drift), disabling the use of naive Bayes in splitting tests (Version 1) and enabling the use of naive Bayes and drift detection (Default). For *Led* dataset the figures of tree size refer to the mean of all 45 trees.

5%. These datasets are generated by a stationary distribution. Nevertheless there are signals of false alarms drift detection. They never appear in the root node but in deeper nodes in the tree. The impact in performance is reduced or even null.

4.2 Non-Stationary Data

For illustrative purposes we evaluate UFFT in the SEA concepts, previously used in [19] to evaluate the ability to detect concept drift. Table 2(a) presents the average error-rate of 30 runs of UFFT setting on/off the ability of drift detection. The results are different at a significance level of 99%. They clear indicate the benefits of using drift detection in this dataset. For reference we also present the results of CVFDT.

The Electricity Market Dataset was collected from the Australian NSW Elec-

(a)Sea Dataset

	UFFT		CVFDT
	Drift	No Drift	
Mean	12.79	17.20	14.72
Variance	1.26	1.89	1.06

(b)Electricity market Dataset

	Lower	*Upper Bound*		UFFT
Test Set	*Bound*	*All Data*	*Last Year*	
Last Day	8.0	20.8	12.5	10.4
Last Week	14.5	25.0	24.7	21.4

Table 2: Error-rates on drift detection problems.

tricity 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 class label identifies the change of the price related to a moving average of the last 24 hours. The goal of the problem is to predict if the price will increase or decrease. From the original dataset we design two experiments. In one of the experiments, the test set is the last day (48 examples); in the other, the test set is the last week (336 examples). For each problem, we detect a lower bound and an upper bound of the error using a batch decision tree learner. The upper bound use ad-hoc heuristics to choose the training set. One heuristic use all the training data; the other heuristic use only the last year training examples. When predicting the last day, the error-rate of Rpart[2] using all the training set is 18.7%, when restricting the training set to the last year, the error decrease to 12.5%. To compute the lower-bound we perform an exhaustive search for the best training set that produces lower error rate in the last day of the training set. The results appear in Table 2(b).

The experiments using UFFT with drift detection exhibit a performance similar to the lower-bound using exhaustive search. This is an indication of the quality of the results. The advantage of using a drift detection method is the ability to automatically choose the set of training examples. This is a real world dataset where we do not know where the context is changing.

5 Conclusions and Future Work

This work presents an incremental learning algorithm appropriate for processing high-speed numerical data streams. The main contributions of this work are the ability to use multivariate splitting tests, and the ability to adapt the decision model to concept drift. While the former has impact in the performance of the system, the latter extends the range of applications to dynamic environments.

The UFFT system can process new examples as they arrive, performing a single scan of the training data. The method to choose the cut point for splitting tests is based on quadratic discriminant analysis. It has complexity

[2] The version of Cart implemented in R [9].

$O(\#examples)$. The sufficient statistics required by the analytical method can be computed in an incremental way, guaranteeing constant time to process each example. This analytical method is restricted to two-class problems. We use a forest of binary trees to solve problems with more than 2 classes. Other contributions of this work are the use of a short-term memory to initialize new leaves, and the use of functional leaves to classify test cases.

An important aspect in this work, is the ability to detect changes in the distribution of the examples. To detect concept drift, we maintain, at each inner node, a naive-Bayes classifier trained with the examples that cross the node. While the distribution of the examples is stationary, the online error of naive-Bayes will decrease. When the distribution changes, the naive-Bayes online error will increase. In that case the test installed at this node is no more appropriate for the actual distribution of the examples. When this occurs all the subtree rooted at this node will be pruned. The pruning corresponds to forget older examples. The empirical evaluation, using stationary data, shows that UFFT is competitive to the state of the art in batch decision tree learning, using much less computational resources. There are two main factors that justifies the overall good performance of the system. One is the use of more powerful classification strategies at tree leaves. The other is the ability to use multivariate splits. The experimental results using non-stationary data, suggest that the system exhibit fast reaction to changes in the concept to learn. The performance of the system indicates that there is a good adaptation of the decision model to the actual distribution of the examples. We should stress that the use of naive-Bayes classifiers at leaves to classify test examples, the use of naive-Bayes as splitting-tests, and the use of naive-Bayes classifiers at decision nodes to detect changes in the distribution of the examples are directly obtained from the sufficient statistics required to compute the splitting criteria, without no additional computations. This aspect is a main advantage in the context of high-speed data streams.

Acknowledgments

This work was developed in the context of projects RETINAE, and ALESII (POSI/EIA/55340/2004).

References

1. C. Blake, E. Keogh, and C. Merz. UCI Repository of Machine Learning, 1999.
2. P. Domingos and G. Hulten. Mining high-speed data streams. In I. Parsa, R. Ramakrishnan, and S. Stolfo, editors, *Proceedings of the Six International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, 2000.
3. R. Duda, P. Hart, and D. Stork. *Pattern Classification.* Willey and Sons, 2001.
4. J. Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

5. J. Gama. Functional trees. *Machine Learning*, 55(3):219–250, 2004.
6. J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In P.Domingos and C. Faloutsos, editors, *9th ACM SigKDD Int. Conference*. ACM Press, 2003.
7. G. Hulten and P. Domingos. Catching up with the data: research issues in mining data streams. In *Proc. of Workshop on Research issues in Data Mining and Knowledge Discovery*, 2001.
8. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In F. Provost, editor, *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2001.
9. R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
10. J. Kittler. Combining classifiers: A theoretical framework. *Pattern analysis and Applications*, 1(1), 1998.
11. R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281 – 300, 2004.
12. R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In P. Langley, editor, *Proceedings of ICML-00, 17th International Conference on Machine Learning*, pages 487–494. Morgan Kaufmann Publishers, San Francisco, US, 2000.
13. R. Klinkenberg and I. Renz. Adaptive information filtering: Learning in the presence of concept drifts. In *Learning for Text Categorization*, pages 33–40. AAAI Press., 1998.
14. M. Kubat and G. Widmer. Adapting to drift in continuous domain. In *Proceedings of the 8th European Conference on Machine Learning*, pages 307–310. Spinger Verlag, 1995.
15. W.-Y. Loh and Y.-S. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7(4), 1997.
16. M. Maloof and R. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41:27–52, 2000.
17. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
18. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.
19. N. Street and Y. Kim. A streaming ensemble algorithm for large-scale classification. In *Proc. 7th ACM SIGKDD*, pages 377–382. ACM Press, 2001.
20. R. Vicente, O. Kinouchi, and N. Caticha. Statistical mechanics of online learning of drifting concepts: a variational approach. *Machine Learning*, 32:179, 1998.
21. H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In P.Domingos and C. Faloutsos, editors, *9th ACM SigKDD Int. Conference*. ACM Press, 2003.
22. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.