Concept Drift in Decision Trees Learning from Data Streams

João Gama, Pedro Medas, and Pedro Rodrigues
LIACC, FEP - University of Porto
Rua Campo Alegre 823, 4150 Porto, Portugal
email: [jgama, pmedas]@liacc.up.pt, prodrigues@dcc.fc.up.pt

ABSTRACT: This paper presents the Ultra Fast Forest of Trees (UFFT) system. It is an incremental algorithm that works online, processing each example in constant time, and performing a single scan over the training examples. The system has been designed for numerical 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. To detect concept drift, we maintain, at each inner node, a naive-Bayes classifier. Statistical theory states that while the distribution of the examples is stationary, the online error of naive-Bayes will decrease, otherwise, the test installed at this node is not appropriate for the actual distribution of the examples. When this occurs, the entire sub tree rooted at this node is pruned. The use of naive-Bayes classifiers at leaves to classify test examples, 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 any additional computations. This aspect is a main advantage in the context of high-speed data streams. The experimental results show a good performance at the change of concept detection and also with learning the new concept.

KEYWORDS: Concept Drift, Forest of Trees, Data Streams.

INTRODUCTION

Decision trees, due to its characteristics, are one of the most used techniques for data-mining. Decision tree models are non-parametric, distribution-free, and robust to the presence of outliers and irrelevant attributes. Tree models have high degree of interpretability. Global and complex decisions can be approximated by a series of simpler and local decisions. Univariate trees are invariant under all (strictly) monotone transformations of the individual input variables. Usual algorithms that construct decision trees from data use a divide and conquer strategy. A complex problem is divided into simpler problems and recursively the same strategy is applied to the sub-problems. The solutions of sub-problems are combined in the form of a tree to yield the solution of the complex problem. Formally a decision tree is a direct acyclic graph in which each node is either a decision node with two or more successors or a leaf node. A decision node has some condition based on attribute values. A leaf node is labelled with a constant that minimizes a given loss function. In the classification setting, the constant that minimizes the 0-1 loss function is the mode of the class of the examples that reach this leaf.

Data streams are problems where the training examples used to construct decision models come over time, usually one at time. In usual real-world applications the data flows continuously at high-speed. In these situations is highly improvable 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. Natural approaches for these incremental tasks are adaptive learning algorithms, that is, incremental learning algorithms that take into account concept drift. In this paper we present UFFT, an algorithm that generates forest-of-trees for data streams. The main contributions of this work are a fast method, based on discriminant analysis, to choose the cut point for splitting tests, the use of a short-term memory to initialize new leaves, 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 used in the functional leaves 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. The preliminary results are presented in Section 4. In the last section we summarize the main contributions of this paper, and point out some future work.
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. Other works related to the work presented here include the use of more powerful classification strategies at tree leaves, and the use of forest of trees.

In the literature of machine learning, several methods have been presented to deal with time changing concepts [13][17][11][12][8]. The two basic methods are based on temporal windows where the window fixes the training set for the learning algorithm and weighting examples which 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. Older data will 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 [17]. 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 [17] and Klinkenberg [11] 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 [13] describe a system that adapts to drift in continuous domains. Klinkenberg [10] 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.

G. Hulten and P. Domingos have proposed a method to scale-up learning algorithms to very-large databases [1][7]. They have presented system VFDT [1], 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. The CVFDT [8] is an algorithm for mining decision trees from continuous-changing data streams. At each node CVFDT maintains the sufficient statistics to compute the splitting-test. Each time an example traverses the node the statistics are updated. After seeing \( n_{min} \) examples, 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. This is a fundamental difference to our method. The CVFDT uses old data to maintain two sub trees. The UFFT with drift detection prunes the sub tree when detecting a new context and only uses the current example. The CVFDT constantly verifies the validity of the splitting test. The UFFT only verifies that the examples at each node have a stationary distribution function.

ULTRA-FAST FOREST TREES - UFFT

The UFFT [4] is an algorithm for supervised classification learning that generates a forest of binary trees. The algorithm is incremental, processing each example in constant time, works on-line, and uses the Hoeffding bound to decide when to install a splitting test in a leaf leading to a decision node. 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 problems and large and medium datasets. 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 [2] trained with the examples that traverse the node. Statistical theory guarantees that for stable distribution of the examples, the online error of naive-Bayes will decrease. When the distribution 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.

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 \( \text{attribute} < \text{value} \). We use the analytical method for split selection presented in [14]. We choose, for all numerical attributes, the most promising value \( \text{value} \). 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 [16] and in VFDTc [5], 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. Let \( \phi(\mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \) be the normal density function, where \( \mu \) and \( \sigma^2 \) 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 \((-\infty, d_1) \), \((d_1, d_2) \), \((d_2, \infty) \) where \( d_1 \) and \( d_2 \) are the possible roots of the equation \( p(-)\phi(\mu_1, \sigma_1) = p(+)\phi(\mu_2, \sigma_2) \) 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 \( \text{Att}_i \leq d \). 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 \( d \). 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 \( d \). 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 and incrementally maintained. In [5] the authors presented an extension to VFDT to deal with continuous attributes. They use a Btree to store continuous attribute-values with complexity \( O(n \log(n)) \). The complexity of the proposed method 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.

FROM LEAF TO DECISION NODE

To expand the tree, a test \( \text{attribute} \leq d \) 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 favour of the best splitting test, is asserted using the Hoeffding bound as in VFDT [1].

An innovation in UFFT is the method used to determine how many examples are needed to evaluate the splitting criteria. As we have pointed out, a new instance does not automatically triggers the splitting decision criteria. Only after receiving \( n_{\text{min}} \) examples since the last evaluation instances will the algorithm execute the splitting test criteria. The

\footnote{1 The reader should note that in UFFT any \( n \)-class problem is decomposed into \( n(n-1)/2 \) two-class problems.}
number of examples needed till the next evaluation is computed as \((1.0/(2*\delta))^{*}\log(2.0/\varepsilon)\), with confidence 
\(1-\delta\) and \(\varepsilon\) easily derived from the Hoeffding bounds.

When new nodes are created, the short term memory is used. This is described in the following section.

**SHORT TERM MEMORY**

When a new leaf is created its sufficient statistics are initialized to zero. The new leaves are trained with the past examples that are in the new concept. We use a short term memory that maintains a limited number of the most recent examples. This short term memory is used to update the statistics at new leaves when they are created. The examples in the short term memory traverse the tree. The ones that reach the new leaves will update the sufficient statistics of these leaves. 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.

**CLASSIFICATION STRATEGIES AT LEAVES**

To classify an unlabelled 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 \(P(x_i \mid \text{Class})\) 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: 
\[
P(C^i \mid x) \propto \log(\Pr(C^i)) + \sum_{k=1}^{l} \log(\phi(\bar{x}_k^i, s_k^i)).
\]

There is a simple motivation for this option. UFFT only changes a leaf to a decision node when there are 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 uses 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 [5].

**FOREST OF TREES**

The splitting criterion only applies to two class problems. In the original paper [14] 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 superclasses. 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. 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 [3] 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 \(n(n-1)/2\) 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.

**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 registered. Each of the trees 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 [9]. 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. This is because each classifier must label all examples as belonging to one of the two classes it was trained on.

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 [15]. When the system detects an increase of the naive-Bayes error in a given node, an indication of a change in the distribution of the examples, this suggests that the splitting-test that has been installed at this node is no longer appropriate. In such cases, the entire sub tree 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 stored in the short term memory. We designate as context a set of contiguous examples where the distribution is stationary. We assume that the data stream is a set of contexts. The goal of the proposed 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 code till a leaf. At each node, the naive Bayes installed at that node classify 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 $p_i \equiv \frac{\text{error}_i}{i}$ 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

$$s_i = \sqrt{\frac{p_i (1-p_i)}{i}}$$

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 $p \pm \alpha s_i$. The parameter $\alpha$ depends on the confidence level. In our experiments, the confidence level has been set to 99%. The drift detection method manages two registers during the training of the learning algorithm, $p_{\text{min}}$ and $s_{\text{min}}$. Every time a new example $i$ is processed those values are updated when $p_i + s_i$ is lower or equal than $p_{\text{min}} + s_{\text{min}}$.

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 $p_i + s_i \geq p_{\text{min}} + 1.5 s_{\text{min}}$. The drift level is reached if $p_i + s_i \geq p_{\text{min}} + 3 s_{\text{min}}$. 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 indication 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 example in the short term memory whose timestamp is greater than $k_d$. It is possible to observe an increase of the error reaching the warning level, followed by a decrease. We assume that such situations correspond to a false alarm, without changing the context. Fig. 1 details the time window structure. 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.
EXPERIMENTAL WORK

ARTIFICIAL DATA

The experimental work was done using artificial data previously used in [13] and a real-world problem described in [6]. The artificial problems allow us to perform controlled experiments, choosing the point of drift, the form of changing drift and verifying when and how the algorithm reacts. In the real-world problem, we do not know if and when there is drift.

One dataset has a gradual context drift. The other has an abrupt context drift. Both datasets have 2 contexts, having 10000 examples in each context. Each dataset is composed of 20000 random generated examples. The test datasets have 1000 examples from the last context in the train dataset. In the first dataset, the examples are uniformly distributed in a 2-dimensional unit square. In the first concept, all points below the curve $y=\sin(x)$ are positive and the remainder are negative. A change in the context corresponds reverses the class label, that is abrupt concept drift. In the second dataset, points inside a circle are positive and outside the circle are negative. The circles move slowly along a line that is a gradual change of the context. These characteristics allow us to assess the performance of the method in various conditions. The results in term of error-rate in the test set are presented in Fig. 2. They are averages of runs in 30 randomly generated datasets. The results show a significantly advantage for using UFFT with context drift detection versus UFFT with no drift detection. For both datasets, the results obtained for the UFFT with drift detection shown no overhead on the processing time and no need of more computational resources.

THE ELECTRICITY MARKET DATASET

The data was collected from the Australian NSW 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 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 (Rpart - the version of Cart implemented in R [18]). 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 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 on Fig. 2. The experiment using UFFT with drift detection exhibits 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 choose the set of training examples automatically. This is a real world dataset where we do not know where the context is changing.
CONCLUSIONS AND FUTURE WORK

This work presents an incremental learning algorithm appropriate for processing high-speed numerical data streams with the capability to adapt to concept drift. The UFFT system can process new examples as they arrive, using a single scan of the training data. The method to choose the cut point for splitting tests is based on discriminant analysis. It has complexity $O(\#\text{examples})$. 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, the use of functional leaves to classify test cases, and the use of a dynamic method to decide how many examples are needed to evaluate candidate splits. 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 not appropriate for the actual distribution of the examples. When this occurs the entire sub tree rooted at this node is pruned. The pruning corresponds to forgetting older examples. 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, 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 any additional computations. This aspect is a main advantage in the context of high-speed data streams.

ACKNOWLEDGMENTS

The authors reveal its gratitude to the financial support given by the FEDER, and the Plurianual support attributed to LIACC. This work was developed in the context of the project ALES (POSI/SRI/39770/2001).

REFERENCES


