Accurate decision trees for mining high-speed data streams

Conference Paper · January 2003

DOI: 10.1145/956750.956813 · Source: DBLP

3 authors, including:

João Gama
University of Porto
383 PUBLICATIONS **7,626** CITATIONS

SEE PROFILE

Accurate Decision Trees for Mining High-speed Data Streams

João Gama
LIACC, FEP, Univ. do Porto
R. do Campo Alegre 823
4150 Porto, Portugal
jgama@liacc.up.pt

Ricardo Rocha
Projecto Matemática Ensino
Departamento de Matemática
3810 Aveiro, Portugal
ricardor@mat.ua.pt

Pedro Medas
LIACC, Univ. do Porto
R. do Campo Alegre 823
4150 Porto, Portugal
pmedas@liacc.up.pt

ABSTRACT

In this paper we study the problem of constructing accurate decision tree models from data streams. Data streams are incremental tasks that require incremental, online, and any-time learning algorithms. One of the most successful algorithms for mining data streams is VFDT. In this paper we extend the VFDT system in two directions: the ability to deal with continuous data and the use of more powerful classification techniques at tree leaves. The proposed system, VFDTc, can incorporate and classify new information online, with a single scan of the data, in time constant per example. The most relevant property of our system is the ability to obtain a performance similar to a standard decision tree algorithm even for medium size datasets. This is relevant due to the any-time property. We study the behaviour of VFDTc in different problems and demonstrate its utility in large and medium data sets. Under a bias-variance analysis we observe that VFDTc in comparison to C4.5 is able to reduce the variance component.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*data mining*; I.2.6 [**Artificial Intelligence**]: Learning—*classifiers design and evaluation*

Keywords

Data Streams, Incremental Decision Trees, Functional Leaves

1. INTRODUCTION

Databases are rich in information that can be used in the decision process. Nowadays, most of the companies and organizations possess gigantic databases, that grow to a limit of millions of registers per day. In traditional applications of data mining the volume of data is the main obstacle to the use of memory-based techniques due the restrictions in the computational resources: memory, time or space in hard disk. Therefore in most of these systems, the use of all available data becomes impossible and can result in under fitting. The construction of KDD systems that use the entire amount of data and keep the accuracy of the traditional systems becomes problematic.

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.

Data streams are, by definition, problems where the training examples used to construct decision models come over time, usually one at time. A natural approach for this *incremental task* is *incremental learning algorithms*. In the field of incremental tree induction a successful technique maintains at each decision node a set of sufficient statistics and only make a decision (install a split-test in that node), when there is enough statistical evidence in favour of a particular split test. This is the case of [6, 3]. In this paper we argue that incremental tree induction methods that only install a split-test when there is enough statistical support, will large benefit of using more appropriate classification strategies at tree leaves. This is the main idea behind this paper.

We propose the VFDTc system, which incorporates two main extensions to the VFDT system: the ability to deal with numerical attributes and the ability to apply naive Bayes classifiers in tree leaves. The paper is organized as follows. The next section describes VFDT and other related work that is the basis for this paper. The section 3 presents our extensions to VFDT leading to system VFDTc. We detail the major options that we implemented and the differences to VFDT and others available systems. The system has been implemented and evaluated. Experimental evaluation is done in Section 4. Last section concludes the paper, resuming the main contributions of this work.

2. RELATED WORK

In this section we analyse related work in two dimensions.

SIGKDD '03, August 24-27, 2003, Washington, DC, USA

One dimension is related to the use of more powerful classification strategies at tree leaves, the other dimension is related to methods for incremental tree induction.

Functional Tree Leaves

The standard algorithm to construct a decision tree usually install at each leaf 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 target attribute of the examples that fall at this leaf. Several authors have studied the use of other functions at tree leaves [9, 5]. One of the earliest works is the Perceptron tree algorithm [11] where leaf nodes may implement a general linear discriminant function. Also Kohavi [9] has presented the naive Bayes tree that uses functional leaves. NBtree is a hybrid algorithm that generates a regular univariate decision tree, but the leaves contain a naive Bayes classifier built from the examples that fall at this node. The approach retains the interpretability of naive Bayes and decision trees, while resulting in classifiers that frequently outperform both constituents, especially in large datasets.

In this paper we explore this idea in the context of learning from data streams. As we show in the experimental section, there are strong advantages in the performance of resulting decision models.

Incremental Tree Induction

In many interesting domains, the information required to learn concepts is rarely available a priori. Over time, new pieces of information become available, and decision structures should be revised. This learning mode has been identified and studied in the machine learning community under several designations: *incremental learning*, *online learning*, *sequential learning*, *theory revision*, etc. In the case of tree models, we can distinguish two main research lines. The first one, a tree is constructed using a greedy search. Incorporation of new information involves the re-structuring the actual tree. This is the case of systems like ITI [12], or ID5R[8]. The second research line does not use the greedy search of standard tree induction. It maintains a set of sufficient statistics at each decision node and only make a decision, i. e., install a split-test at that node when there is enough statistical evidence in favour of a split test. This is the case of [6, 3]. A notable example, is the VFDT system [3]. It can manage thousand of examples using few computational resources with a performance similar to a batch decision tree given enough examples.

In VFDT a decision tree is learned by recursively replacing leaves with decision nodes. Each leaf stores the sufficient statistics about attribute-values. The sufficient statistics are those needed by a heuristic evaluation function that evaluates the merit of split-tests based on attribute-values. When an example is available, it traverses the tree from the root to a leaf, evaluating the appropriate attribute at each node, and following the branch corresponding to the attribute's value in the example. When the example reaches a leaf, the sufficient statistics are updated. Then, each possible condition based on attribute-values is evaluated. If there is enough statistical support in favour of one test over the others, the leaf is changed to a decision node. The new decision node will have as many descendant leaves as the number of possible values for the chosen attribute (therefore this tree is not necessarily binary). The decision nodes only maintain the information about the split-test installed in this node. The initial state of the tree consists of a single leaf: the root of the tree. The heuristic evaluation function is the Information Gain (denoted by $H(\cdot)$)[1]. The sufficient statistics for estimating the merit of a nominal attribute are the counts n_{ijk}, representing the number of examples of class k that reach the leaf, where the attribute j takes the value i. The Information Gain measures the amount of information that is necessary to classify an example that reach the node: $H(A_j) = info(examples) - info(A_j)$. The information of the attribute j is given by: $info(A_j) = \sum_i P_i \left(\sum_k -P_{ik} \log_2(P_{ik}) \right)$ where $P_{ik} = \frac{n_{ijk}}{\sum_a n_{ajk}}$, is the probability of observing the value of the attribute i given class k and $P_i = \frac{\sum_a n_{ija}}{\sum_a \sum_b n_{ajb}}$ is the probability of observing the value of attribute i.

The main innovation of the VFDT system is the use of Hoeffding bounds to decide how many examples are necessary to observe before installing a split-test at each leaf. Suppose we have made n independent observations of a random variable r whose range is R. The Hoeffding bound states, with probability $1-\delta$, that the true average of r, $\bar{r}$, is at least $\bar{r} - \epsilon$ and $\epsilon = \sqrt{R^2 \frac{\ln(\frac{1}{\delta})}{2n}}$.

Let $H(\cdot)$ be the evaluation function of an attribute. For the information gain, the range R, of $H(\cdot)$ is $log_2 \#classes$. Let x_a be the attribute with the highest $H(\cdot)$, x_b the attribute with second-highest $H(\cdot)$ and $\overline{\Delta H} = \overline{H}(x_a) - \overline{H}(x_b)$, the difference between the two better attributes. Then if $\overline{\Delta H} > \epsilon$ with n examples observed in the leaf, the Hoeffding bound states with probability $1-\delta$ that x_a is really the attribute with highest value in the evaluation function. In this case the leaf must be transformed into a decision node that splits on x_a.

The evaluation of the merit function for each example could be very expensive. It turns out that it is not efficient to compute $H(\cdot)$ every time that an example arrives. VFDT only computes the attribute evaluation function $H(\cdot)$ when a minimum number of examples has been observed since the last evaluation. This minimum number of examples is a user-defined parameter. When two or more attributes continuously have very similar values of $H(\cdot)$, even with a large number of examples, the Hoeffding bound will not decide between them. To solve this problem the VFDT uses a constant τ introduced by the user for run-off, e.g., if $\overline{\Delta H} < \epsilon < \tau$ then the leaf is transformed into a decision node. The split test is based on the best attribute.

3. THE VFDTc SYSTEM

We implement a system based on the VFDT[3]. It uses the Information Gain as the evaluation function, deals with numerical attributes and uses functional leaves to classify test examples.

Numerical attributes

Most real-world problems contain numerical attributes. Practical applications of learning algorithms to real-world problems should address this issue. For batch decision tree learners, this ability requires a sort operation that is the most

[1]The original description of VFDT is general enough for other evaluation functions (e.g. GINI). Without loss of generality, we restrict here to the information gain.

Figure 1: Algorithm to insert value x_j of an example label with class y into a Binary Tree

```
Procedure InsertValueBtree(x_j,y,Btree)
  Begin
  if (x_j == Btree.i) then
      Btree.VE[y]++.
  Elseif (x_j ≤ Btree.i) then
      Btree.VE[y]++.
      InsertValueBtree(x_j,y,Btree.Left).
  Elseif (x_j > Btree.i) then
      Btree.VH[y]++.
      InsertValueBtree(x_j,y,Btree.Right).
End .
```

Figure 2: Algorithm to compute $\#(A_j \leq i)$ for a given attribute j and class k:

```
Procedure LessThan(i,k,BTree)
  Begin
  if (BTree == NULL) return 0.
  if (BTree.i == i) return VE[k].
  if (BTree.i < i) return
      VE[k] + LessThan(i, k, BTree.Right).
  if (BTree.i>i)
      return LessThan(i,k,BTree.Left).
End .
```

time consuming operation.In this section we provide an efficient method to deal with numerical attributes in the context of online decision tree learning.

In VFDTc a decision node that contains a split-test based on a continuous attribute has two descendant branches. The split-test is a condition of the form $attr_i \leq cut\_point$. The two descendant branches corresponds to the values $TRUE$ and $FALSE$ for the split-test. The cut_point is chosen from all the possible observed values for that attribute. In order to evaluate the goodness of a split, we need to compute the class distribution of the examples at which the attribute-value is less than or greater than the cut_point. The counts n_{ijk} are fundamental for compute all necessary statistics. They are kept with the use of the following data structure: In each leaf of the decision tree we maintain a vector of the classes distribution of the examples that reach this leaf. For each continuous attribute j, the system maintains a binary tree structure. A node in the binary tree is identified with a value i (that is the value of the attribute j seen in an example), and two vectors (of dimension k) used to count the values that go through that node. These vectors, VE and VH contain the counts of values respectively $\leq i$ and $> i$ for the examples labelled with class k. When an example reaches a leaf, all the binary trees are updated. Figure 1 presents the algorithm to insert a value in the binary tree. Insertion of a new value in this structure is $O(log\ n)$ where n represents the number of distinct values for the attribute seen so far.

To obtain the Information Gain of a given attribute we use an exhaustive method to evaluate the merit of all possible cut_points. In our case, any value observed in the examples so far can be used as cut_point.

For each possible cut_point, we compute the information of the two partitions using equation 1.

$$info(A_j(i)) =$$

$$P(A_j \leq i) * iLow(A_j(i)) + P(A_j > i) * iHigh(A_j(i)) \quad (1)$$

where i is the split point, $iLow(A_j(i))$ the information of $A_j \leq i$ (equation 2) and $iHigh(A_j(i))$ (equation 3) the information of $A_j > i$. So we choose the split point that minimizes (1).

$$iLow(A_j(i)) =$$

$$-\sum_K P(K = k|A_j \leq i) * log_2(P(K = k|A_j \leq i)) \quad (2)$$

$$iHigh(A_j(i)) =$$

$$-\sum_K P(K = k|A_j > i) * log_2(P(K = k|A_j > i)) \quad (3)$$

These statistics are easily computed using the counts n_{ijk}, and using the algorithm presented in figure 2. For each attribute, it is possible to compute the merit of all possible cut_points traversing the binary tree once. A split point for a numerical attribute is binary. The examples will be divided into two subsets: one representing the True value of the split-test and the other the False value of the test installed at the decision node. VFDTc only considers a possible cut_point if and only if the number of examples in each of the subsets is higher than p_{min}[2] percentage of the total number of examples seen in the node.

Discrete attributes

The VFDTc does not need previous knowledge of all the possible values of a categorical attribute. The sufficient statistics for discrete attributes are counters of the number of occurrences of an observed attribute-value per class. These statistics are enough to compute $H(\cdot)$. When a test on a discrete attribute is installed in a leaf, the leaf becomes a decision node with as many descendant branches as the number of observed distinct values [3] plus a branch that represents other, not yet observed, values and unknown values. Therefore when an example reaches this node with an unknown value for this attribute, the example follows the branch representing other values.

Functional tree leaves

To classify a test example, the example traverses the tree from the root to a leaf. The example is classified with the most representative class of the training examples that fall at that leaf. One of the innovations of our algorithm is the ability to use the naïve Bayes classifiers at tree leaves. That is, a test example is classified with the class that maximizes the posterior probability given by Bayes rule assuming the independence of the attributes given the class.

There is a simple motivation for this option. VFDT 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 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

[2]Where p_{min} is a user defined constant.
[3]Known until the moment.

Figure 3: Algorithm to compute $P(x_j|C_k)$ for numeric attribute x_j and class k at a given leaf.

Procedure PNbc(x_j,k,Btree, X_h, X_l,N_j)
X_h the highest value of x_j observed at the Leaf
X_l the lowest value of x_j observed at the Leaf
N_j the different values of x_j observed at the Leaf
Begin
if (BTree==NULL) return 0
$nintervals = min(10, N_j)$. // number of intervals
$inc = \frac{X_h - X_l}{nintervals}$. // interval range
Let $Counts[nintervals]$ be the number
of examples between intervals
For i=1 to nintervals
$Counts[i] = LessThan(x_l + inc * i, k, BTree)$
If $(i > 1)$ then
Counts[i]=Counts[i]-Counts[i-1]
If $(x_j \leq X_l + inc * i)$ then
return $\frac{Counts[i]}{Leaf.nrExs[k]}$
ElseIf ((i==nintervals) then
return $\frac{Counts[i]}{Leaf.nrExs[k]}$
End

the conditional probabilities of the attribute-values given the class. In this way, there is a much better exploitation of the available information at each leaf. Moreover, naive Bayes is naturally incremental. It deals with heterogeneous data and missing values. It has been observed [4] that for small datasets naive Bayes is a very competitive algorithm.

Given the example $\overrightarrow{e} = (x_1, ..., x_j)$ and applying Bayes theorem, we obtain: $P(C_k|\overrightarrow{e}) \propto \frac{P(C_k)}{P(\overrightarrow{e})} \prod P(x_j|C_k)$. To compute the conditional probabilities $P(x_j|C_k)$ we should distinguish between nominal attributes and continuous ones. In the former the problem is trivial using the sufficient statistics used to compute information gain. In the latter, there are two usual approaches: or assuming that each attribute follows a *normal distribution* or discretizing the attributes. Assuming a normal distribution, the sufficient statistics can be computed on the fly. Nevertheless, it is possible to compute the required statistics from the binary-tree structure stored at each leaf before it comes a decision node. This is the method implemented in VFDTc. Any numerical attribute is discretized into $min(10, Nr.\ of\ different\ values)$ intervals. To count the number of examples per class that fall at each interval we use the algorithm described in figure 3. This algorithm is computed only once in each leaf for each discretization bin. Those counts are used to estimate $P(x_j|C_k)$.

We should note, that the use of naive Bayes classifiers at tree leaves doesn't introduce any overhead in the training phase. In the application phase and for nominal attributes, the sufficient statistics constitute (directly) the naive Bayes tables. For continuous attributes, the naive Bayes contingency tables are efficiently derived from the Btree's used to store the numeric attribute-values. The overhead introduced is proportional to depth of the Btree, that is at most $log(n)$, where n is the number of different values observed for a given attribute.

4. EVALUATION

In this section we empirically evaluate VFDTc. We consider three dimensions of analysis: error rate, learning time, and tree size. The main goal of this section is to provide evidence that the use of functional leaves improve the performance of VFDT and most important that it improve its *anytime* characteristic. In a first set of experiments we analyze the effects of two different strategies when classifying test examples: classifying using the majority class (VFDTcMC) and classifying using naive Bayes (VFDTcNB) at leaves.

The experimental work has been done using the *Waveform* and *LED* datasets. These are well known artificial datasets. We have used the two versions of the *Waveform* dataset available at the UCI repository [1]. Both versions are problems with three classes. The first version 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 choice of these datasets was motivated by the existence of dataset generators at the UCI repository. We could generate datasets with any number of examples and perform a set of learning curves able to evaluate our claim about the *any-time* property of VFDTc.

We have done a set of experiments, using the *LED* and *Waveform* datasets. For all datasets, we generate training sets of a varying number of examples, starting from 10k till 1000k. The test set contains 250k examples. The VFDTc algorithm was used with the parameters values $\delta = 5 \times 10e^{-6}, \tau = 5 \times 10e^{-3}$ and $n_{min} = 200$.

All algorithms run on a Pentium IV at 1GHz with 256 MB of RAM and using Linux RedHat 7.1. Detailed results are presented in table 1.

Classi£cation strategy: Majority Class vs. naive Bayes

In this subsection, we compare the performance of VFDTc using two different classification strategies at the leaves: naive Bayes and Majority Class. Our goal is to show that using stronger classification strategies at tree leaves will improve, sometimes drastically, classifier's performance.

On these datasets VFDTcNB consistently out-performs VFDTcMC (Figure 4). We should note, with the *Waveform* data, as the number of examples increases, the performance of VFDTcMC approximates VFDTcNB (Figure 4). The most relevant aspect in all learning curves is that the performance of VFDTcNB is almost constant, independently of the number of training examples. For example the difference between the best and worst performance (over all the experiments) is:

	C4.5	VFDTcNB	VFDTcMC
Waveform (21 atts)	4.28	2.89	18.54
Waveform (40 atts)	3,97	5	13,65
Led	0.41	0.3	7.97

These experiments support our claim that the use of appropriate classification schemes will improve the *any time* property.

With respect to the other dimensions of analysis, the size of the tree does not depend on the classification strategy. With respect to the learning times, the use of naive Bayes classifiers introduces an overhead. The overhead is due to two factors. The first factor only applies when there are numeric attributes and is related to the construction of the contingency tables from the Btrees. The size of these tables is usually short (in our case $10 \times \#Classes$) and independent of the number of examples. In our experiments it

	Error Rate (%)			Learning Times (seconds)				Tree Size	
				Training+Classification			Training		
Nr. Exs	**C4.5**	**VFDTcNB**	**VFDTcMC**	**C4.5**	**VFDTcNB**	**VFDTcMC**	**VFDTc**	**C4.5**	**VFDTcNB**
Waveform dataset - 21 Attributes									
100k	19.98	17.85	25.46	229.2	31.2	23.4	18.3	7901	51
500k	18.96	17.28	21.15	3975.1	116.5	106.9	101.5	37787	249
1000k	18.17	17.15	20.16	14960.8	210.1	201.5	195.3	61323	495
Waveform dataset - 40 Attributes									
100k	21.02	18.50	24.82	347.8	64.2	46.2	36.2	9935	51
500k	19.63	17.17	21.29	5570.8	227.4	155.3	195.1	43725	243
1000k	19.12	17.03	19.82	22402.7	431.3	408.2	397.4	80933	491
LED dataset - 24 Attributes									
100k	26.53	25.87	74.67	15.1	18.8	6.3	2.2	7923	39
500k	26.39	25.95	73.97	80.6	31.7	19.9	14.1	35413	207
1000k	26.35	26.05	73.98	190.5	59.9	45.3	36.8	65119	403

Table 1: Learning curves for Waveform-21, Waveform-40, and LED datasets. In all the experiments the test set have 250k examples.

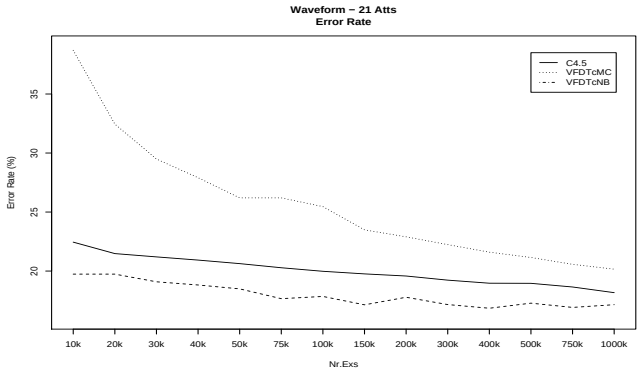

Figure 4: Learning Curves of VFDTcMC, VFDTcNB and C4.5 error-rates on Waveform data (21 atts).

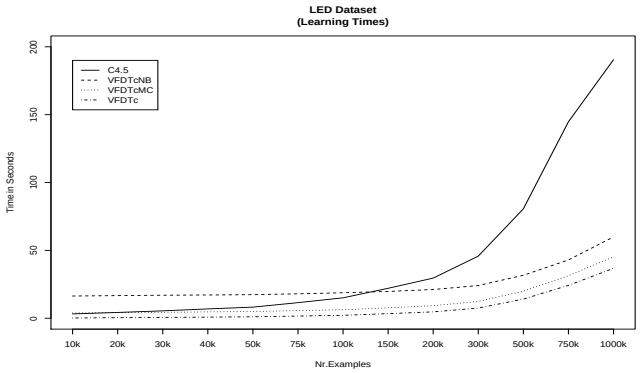

Figure 5: Learning times (train+test) of C4.5, VFDTcNB, and VFDTcMC as a function of the number of training examples.

was the least important factor. The second factor is that the application of naive Bayes requires the estimation, for each test example, of $\#Classes \times \#Attributes$ probabilities. The majority class strategy only requires $\#Classes$ probabilities. When the number of test cases is large (as in our experiments) this is the most relevant factor. Nevertheless the impact of the overhead shrinks has the training time increases. It is why the overhead is more visible for reduced number of training examples (Figure 5).

From now on we focus our analysis in VFDTcNB.

C4.5 versus VFDTcNB

In this subsection, we compare VFDTcNB against C4.5 [10]. VFDTc as VFDT was designed for fast induction of interpretable, and accurate models from large data streams using one scan of the data. The motivation for these experiments is the comparison of the relative performances of an online learning algorithm with a standard batch learner. We would expect, given enough examples, a faster convergence rate of VFDTcNB in comparison to VFDTcMC. The following table shows the mean of the ratios of the error rate (VFDTcNB / C4.5) and the $p-value$ of the Wilcoxon test for the three datasets under study.

	VFDTcNB / C4.5	$p-value$
Waveform (21 atts)	0.89	0.0011
Waveform (40 atts)	0.90	0.0001
LED	0.98	0.0017

On the *LED* dataset the performance of both systems are quite similar unrespectable to the dimension of the training set. On both *Waveform* VFDTcNB out-performs C4.5.

Tree size and Learning Times

In this work, we measure the size of tree models as the number of decision nodes plus the number of leaves. We should note that VFDTcNB and VFDTcMC generates exactly the same tree model. In all the experiments we have done, VFDTc generates decision models that are, at least, one order of magnitude smaller than those generated by C4.5. The size of C4.5 trees grows much more with the number of examples, just as it would expect.

In another dimension, we measured the time needed by each algorithm to generate a decision model. The analysis we have done in a previous section, comparing VFDTcNB versus VFDTcMC applies in the comparison VFDTcNB versus C4.5. VFDTcNB is very fast in the training phase. It scans the entire training set once, and the time needed to process each example is negligible. In the application phase

there is an overhead due to the use of naive Bayes at leaves. In Figure 5 we plot the learning plus classification time as a function of the number of training examples. For small datasets (less than 100k examples) the overhead introduced in the application phase is the most important factor.

Bias-Variance Decomposition of the Error

An interesting analysis of the classification error is given by the so-called *Bias-Variance* decomposition [2]. Several authors refer that there is a trade-off between the systematic errors due to the representational language used by an algorithm (the *bias*) and the *variance* due to the dependence of the model to the training set. We have used the *Waveform* (21 attributes) dataset. The experimental methodology was as follows: We generate a test set with 50k examples, and 10 independent training sets with 75k. VFDTc and C4.5 learn the training sets and the corresponding models were used to classify the test set. The predictions are used to compute the terms of the *bias* and *variance* equations using the definition presented in [2]. The figures of bias and variance for C4.5 were 15.01 and 4.9 respectively, and for VFDTcNB the bias is 17.2 and variance 2.4. We observe that while VFDTc exhibits lower variance, C4.5 exhibits lower Bias. With respect to the variance, these results were the expected. Decision nodes in VFDTc should be much more stable than greedy decisions. Moreover, naive Bayes is known to have low variance. With respect to the bias component, these results are somewhat surprising. They indicate that sub-optimal decisions could contribute to a bias reduction.

Other Results on Real data

We have done some experiments on real data. We have used the *Forest CoverType* dataset from the *UCI KDD archive*. The goal is to predict the forest cover type from cartographic variables. The problem is defined by 54 variables of different types: continuous and categorical. The dataset contains 581012 examples. Published results for this dataset, using the first 15120 examples for training and the reminder 565892 for test are: 70% accuracy for backpropagation, 58% for a linear discriminant, and 68.6% for C4.5. Using the same training and test sets, the accuracy of VFDTcNB [4] is 62.4% and 34.2% for VFDTcMC.

5. CONCLUSIONS

In this paper, we propose two major extensions to VFDT, one of the most promising algorithms for tree induction from data streams. The first one is the ability to deal with numerical attributes. The second one is the ability to apply naive Bayes classifiers in tree leaves. While the former extends the domain of applicability of the algorithm to heterogeneous data, the latter reinforces the *any-time* characteristic, an important property for any learning algorithm for data streams. We should note that the extensions we propose are integrated. In the training phase only the sufficient statistics required to compute the information gain are stored at each leaf. In the application phase, and for nominal attributes, the sufficient statistics constitute (directly) the naive Bayes tables. For continuous attributes, naive Bayes tables are efficiently derived from the Btree used to store the numeric attribute-values. Nevertheless the application of naive Bayes introduces an overhead with respect to the use of the majority class because the former requires the estimation of much more probabilities than the latter. VFDTc maintains all the desirable properties of VFDT. It is an incremental algorithm, new examples can be incorporated as they arrive, it works online, only see one example once, and using a small processing time per example. The experimental evaluation of VFDTc clear illustrates the advantages of using more powerful classification techniques. In the datasets under study VFDTcNB is a very competitive algorithm even in comparison against the state-of-the art in batch decision tree induction. The bias-variance analysis shows that VFDTcNB generates very stable predictive models concerning variations of the training set.

In this paper, we do not discuss the problem of time-changing concepts [7]. Nevertheless, our extensions could be applied to any strategy that takes into account *concept drift*.

Acknowledgments: The authors reveal its gratitude to the financial support given by the FEDER (Plurianual support attributed to LIACC), and to project ALES.

[4]Results obtained with $\delta = 5 \times 10^{-6}$, $\tau = 5 \times 10^{-3}$ and $n_{min} = 200$.

6. REFERENCES

[1] C. Blake, E. Keogh, and C. Merz. UCI repository of Machine Learning databases, 1999.

[2] P. Domingos. A unified bias-variance decomposition and its applications. In P. Langley, editor, *Machine Learning, Proc. of the 17th International Conference*. Morgan Kaufmann, 2000.

[3] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[4] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–129, 1997.

[5] J. Gama. An analysis of functional trees. In C. Sammut, editor, *Machine Learning, Proc. of the 19th Int. Conference*. Morgan Kaufmann, 2002.

[6] J. Gratch. Sequential inductive learning. In *Proc. of Thirteenth National Conference on Artificial Intelligence*, volume 1, pages 779–786, 1996.

[7] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Knowledge Discovery and Data Mining*, 2001.

[8] D. Kalles and T. Morris. Efficient incremental induction of decision trees. *Machine Learning*, 24(3):231–242, 1996.

[9] R. Kohavi. Scaling up the accuracy of naive Bayes classifiers: a decision tree hybrid. In *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[10] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.

[11] P. Utgoff. Perceptron trees - a case study in hybrid concept representation. In *Proc. of the Seventh National Conference on Artificial Intelligence*. Morgan Kaufmann, 1988.

[12] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.