Forest Trees for On-line Data

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

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

Ricardo Rocha
Campus Universitário de
Santiago
3810-193 Aveiro, Portugal
ricardor@mat.ua.pt

ABSTRACT

This paper presents an hybrid adaptive system for induction of forest of trees from data streams. The Ultra Fast Forest Tree system (UFFT) is an incremental algorithm, with constant time for processing each example, works online, and uses the Hoeffding bound to decide when to install a splitting test in a leaf leading to a decision node. Our system has been 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. The number of examples required to evaluate the splitting criteria is sound, based on the Hoeffding bound. 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 fixed 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 these leaves are initialized with the examples in the short term memory that will fall at these leaves. We study the behavior of UFFT in different problems. The experimental results shows that UFFT is competitive against a batch decision tree learner in large and medium datasets.

Keywords: Hybrid Forest of Trees, Data Streams.

1. 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 classes that reach this leaf.

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 *adaptive learning algorithms*. 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, and the use of functional leaves to classify test cases. The paper is organized as follows. In the next section we present related work in the area of incremental decision-tree induction. 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 resume the main contributions of this paper, and point out some future work.

SAC'04, March 14-17, 2004, Nicosia, Cyprus

2. RELATED WORK

In this section we analyze related work in three dimensions. One dimension is related to the use of more power classification strategies at tree leaves, other dimension is related to methods for incremental tree induction, the other dimension is related to the use of forest of trees.

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 [10, 5]. One of the earliest works is the Perceptron tree algorithm [14] where leaf nodes may implement a general linear discriminant function. Also Kohavi [10] 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.

Adaptive 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*, *adaptive learning*, etc. In the case of tree models, we can distinguish two main research lines. In the first one, a tree is constructed using a greedy search. Incorporation of new information involves the re-structuring of the actual tree. This is done using operators that could pull-up or push-down decision nodes. This is the case of systems like ITI [15], 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 makes 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] and its descendent [7]. It can manage millions of examples using few computational resources with a performance similar to a batch decision tree given enough examples.

The VFDT System. 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, testing 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} = number of examples that reached the leaf with class k, attribute j and value of attribute i. The Information Gain measures the amount of information that it would be necessary to classify an example that reached 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 see before to install a split-test at a given 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, $\bar{r}$ it 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 after seeing n examples. Then if $\overline{\Delta H} > \epsilon$ then 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 an example arrives. VFDT only computes the attribute evaluation function $H(\cdot)$ when a minimum number of examples has been seen since the last evaluation. This minimum number of examples is an user-defined parameter. When two or more attributes continuously have very similar values of $H(\cdot)$, even given 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. If $\overline{\Delta H} < \epsilon < \tau$ then the leaf is transformed into a decision node. The split test is based on the best attribute. It is not efficient to calculate $H(\cdot)$ every time a new example arrives. Moreover, it is not provable that the arrival of a single new example will change $\overline{\Delta H} < \epsilon$. as such, The algorithm uses a constant defined by the user, n_{min}, that is the number of necessary examples to reach a leaf before computing $H(\cdot)$.

Forest of Trees. In the field of combining classifiers several algorithms generate forest of trees. Breiman [2] have presented an algorithm to randomly generate a forest of trees. Each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Breiman shows that using a random selection of features to split each node yields error rates that compare favorably to Adaboost. In our case there is no random factor. The forest of trees is obtained by decomposing a k-class problem into $k(k-1)/2$ binary problems.

3. ULTRA-FAST FOREST TREES - UFFT

We present an algorithm for supervised classification learning, that generates a forest of binary trees. The algorithm is incremental, with constant time for processing each example, works online, and uses the Hoeffding bound to decide when to install a splitting test in a leaf leading to a decision node. Our system, 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

[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.

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 these leaves are initialized with the examples in the short term memory that will fall at these leaves. In the following sections we provide detailed information about the most relevant aspects of our algorithm.

The Splitting Criteria. Any 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 branches to the new leaves corresponds to the values *True* and *False* of the splitting test. All splitting tests are of the form $attribute_i \leq value_j$. We use the analytical method for split point selection presented in [11] to 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, like the exhaustive method used in C4.5 [13] and in VFDTc [7], 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 to process each example.

The analytical method uses a modified form of quadratic discriminant analysis to include different variances on the two classes[2]. 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, 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 of examples at the ode. 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 $p(-)$ denotes the estimated probability than an example belongs to class $-$. We want 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 $Att_i \leq d_i$.

We use the information gain as the heuristic 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 less than d and greater than d:

	$Att_i \leq d_j$	$Att_i > d_j$
Class +	p_1^+	p_2^+
Class −	p_1^-	p_2^-

The information kept on 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 this 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 information gain of an attribute, $H(Att_i)$ is: $H(Att_i) = info(p^+, p^-) + \sum_{j=1}^{2} \left(p_j * info(p_j^+, p_j^-)\right)$ where $info(p^+, p^-) = -p^+ log_2(p^+) - p^- log_2(p^-)$ and $p^- = p_1^- + p_2^-$, $p^+ = p_1^+ + p_2^+$. The splitting test with maximum information gain is chosen. This method only requires to maintain the mean and standard deviation for each class per attribute. Both quantities are easily maintained incrementally. In [7] 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 proposed method 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, the test $Att_i \leq d_i$ is installed in the 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 that 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, that it must exist statistical support in favour of the best splitting test, is asserted using the Hoeffding bound as in VFDT.

The 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_{min} examples since the last evaluation instances will the algorithm execute the splitting test criteria. While in VFDT, n_{min} is a user defined constant, in UFFT, the number of examples needed till the next evaluation is computed as $(1.0/(2 * \delta)) * \log(2.0/\epsilon)$. This expression is easly derived from the Hoeffding bound [12].

Short Term Memory. When a new leaf is created its sufficient statistics are initialized to zero. We want the new leaf to have some memory of the last examples that traversed the tree. We use a short term memory that maintains in memory a limited number of the most recent examples. This short term memory is used to update the statistics at the new leaves when they are created. The examples in the short term memory traverse the tree and the ones 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.

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. This leaf classifies the example. We have implemented two different classification strategies. The simplest classification method is based on the majority class from the training examples that reached that leaf. The example is classified with the most representative class of training examples that reached the leaf. The second classification method uses a *naive-Bayes* classifier. The use of the naive-Bayes classifiers at the tree leaves does not enter any overhead in the training

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

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: $P(C^i|\vec{x}) \propto \log(Pr(C^i)) + \sum_{k=1}^{l} \log(\phi(\bar{x}_k^i, s_k^i))$. 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.

Forest of Trees. The splitting criteria only applies to two class problems. In the original paper [11] 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-classes 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.

Fusion of Classifiers. When doing classification of a test example, the algorithm send 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 because each classifier must label all examples as belonging to one of the two classes it was trained on.

4. EXPERIMENTAL WORK

In this section we empirically evaluate UFFT. In a first set of experiments we analyze the effects of two different strategies when classifying test examples: classifying using the majority class and classifying using naive Bayes at leaves. The experimental work has been done using the *Waveform*, *LED* and *Balance* datasets. These are well known artificial datasets. We have used the two versions of the *Waveform* and *LED* datasets available at the UCI repository [1]. Both versions of *Waveform* 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 *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 1000k. The test set contains 250k examples for the *Waveform* and 100k for the *LED* and *Balance* datasets. 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 the parameters values $\delta = 0.05, \tau = 0.001$ and $n_{min} = 300$. All algorithms run on a AThlon XP2000+ at 166GHz with 512 MB of RAM and using Linux RedHat 7.3. Results are presented in table 1.

Classification Strategy: Majority Class versus naive Bayes. In all the experiments the trend is the reduction of the error-rate when increasing the training set. The results show that the use of stronger classifiers at the leaves strongly increases the performance of the system.

Comparison to C4.5. C4.5 is 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). C4.5 is one of the most successful used tree learning algorithms. 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. On these datasets, the performance of both systems is quite similar. We should note the relative good performance of UFFT using the majority class as classification strategy, for example in comparison to previous results of VFDT. This is main effect of the decomposition of a $n - class$ problem into $n(n-1)/2$ binary problems.

Sensitivity to the Order of the Examples. Several authors [3] refer that decision models built using incremental algorithms are very dependent in the order of the examples. We have analyzed this effect in UFFT. We conducted an experience using the *Waveform* (40 atts) dataset. The training set has 300k examples and the test set 250k examples. We shuffled the training set ten times and measure the error

rate in the test set, and learning time of UFFT. An analysis of the results shows a small variance of the error rate. The variance depends on the classification strategy. The most affected classification strategy is the majority class with a variance of 0.0014. Moreover the learning time is unaffected.

Sensitivity to Noise. We study the behavior of UFFT in the presence of noise. The *Led* dataset generator allows the user to control the level of noise produced in the training set. In our experience the test set was fixed in 100k examples with 10% of noise. The training set was fixed in 200k examples with a noise factor ranging from 0% to 70%.

The analysis of the results of UFFT, shows that til 40% of noise the error rate isn't affected. After 50% error rate increases drastically as expected.

5. CONCLUSIONS

We have presented an incremental learning algorithm appropriate for processing high-speed numerical data streams. The UFFT system can process new examples as they arrive, using a single scan of the training data. The main contribution of this work is a fast method, based on discriminant analysis, to choose the cut point for splitting tests. 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 and sound method to decide how many examples are needed to evaluate candidate splits. The empirical evaluation of our algorithm, clear shows the advantages of using more powerful classification strategies at tree leaves. The performance of UFFT when using naive-Bayes classifiers at tree leaves is competitive to the state of the art in batch decision tree learning, using much less computational resources. There are two main reasons that justifies the overall good performance of our system. The first one, is the use of functional leaves, that guarantees efficient levels of performance at any time. The second reason, is the decomposition of a multi-class problem into binary problems, leading to a forest of binary trees. As future work we are extending the algorithm to detect concept drift in the data stream.

Acknowledgments: The authors reveal its gratitude to the financial support given by the FEDER, the Plurianual support attributed to LIACC, and to the financial contribution of project ALES (POSI/SRI/39770/2001).

	Error Rate			Training Time (seconds)		Tree Size			
	UFFT		C4.5						
Exs	Maj.Cl	NBayes		UFFT	C.4.5	UFTT			C.4.5
Balance dataset - 4 Attributes									
100k	8.5	3.6	3.0	18	41	315	1	1	2929
500k	6.5	3.0	2.1	128	822	1355	1	1	9689
1000k	4.9	2.1	1.7	301	2949	2051	1	1	16223
1500k	4.2	1.8	1.5	517	6246	2535	1	1	21687
Waveform dataset - 21 Attributes									
100k	39.3	20.2	19.9	38	156	11	13	1	7945
500k	26.0	21.2	18.6	223	2734	43	83	65	33787
1000k	40.8	19.3	18.3	409	10802	113	73	1	61841
1500k	37.9	18.3	18.0	642	25202	107	93	1	87077
Waveform dataset - 40 Attributes									
100k	40.1	20.3	21	69	297.62	15	7	1	9837
500k	26.5	20.6	19.5	371	4054	59	53	75	435233
1000k	39.8	19.2	19.1	719	16346	45	49	1	80331
1500k	26.2	20.5	18.9	1157	37917	113	97	57	117435
LED dataset - 24 Attributes									
100k	63.6	26.4	26.7	174	525	30			8309
500k	53.7	26.3	26.7	1372	15209	82			34679
750k	52.8	26.6	26.6	2153	35241	108			50387
1000k	49.7	26.6	26.7	2861	67271	116			65469

Table 1: Learning curves for the datasets under study.

6. REFERENCES

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

[2] Leo Breiman. Random forests. Technical report, University of Berkeley, 2002.

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

[4] J. Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[5] J. Gama. An analysis of functional trees. In *Proc. 19th International Conference Machine Learning*. Morgan Kaufmann Publishers, 2002.

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

[7] J.Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Procs. of the 9th ACM SigKDD Int. Conference in Knowledge Discovery and Data Mining*. ACM Press, 2003.

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

[9] J. Kittler. Combining classifiers: A theoretical framework. *Pattern analysis and Applications, Vol. 1, No. 1*, 1998.

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

[11] W.-Y. Loh and Y.-S. Shih. Split selection methods for classification trees. *Statistica Sinica*, 1997.

[12] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

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

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

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