Mining Data Streams Using Option Trees (revised edition, 2004)

Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{geoff, rkirkby, bernhard}@cs.waikato.ac.nz

Abstract. The data stream model for data mining places harsh restrictions on a learning algorithm. A model must be induced following the briefest interrogation of the data, must use only available memory and must update itself over time within these constraints. Additionally, the model must be able to be used for data mining at any point in time. This paper describes a data stream classification algorithm using an ensemble of option trees. The ensemble of trees is induced by boosting and iteratively combined into a single interpretable model. The algorithm is evaluated using benchmark datasets for accuracy against state-of-the-art algorithms that make use of the entire dataset.

1 Introduction

The volume of data in real-world problems can overwhelm popular machine learning algorithms. They do not scale well with the number of instances and may require more memory than is available. With new data they must re-learn a new model from scratch. Although incremental algorithms have been explored in machine learning, the context for their development was never one of having huge datasets (potentially infinite) and limited memory.

Recently there has been a new focus on algorithms suitable for huge datasets that learn from a single pass over the data, are restricted in how much memory they can use, and can be incrementally updated at a later point in time. Additionally, they should be able to perform their data mining task at any point in time. The data model for such an algorithm is termed a data stream, and streams can be finite or infinite. Algorithms may request a single instance from the stream or may request a buffer of them (sometimes called a *chunk*).

The infinite case, sometimes termed *online*, supposes an endless source of data being continuously generated. Algorithms designed to handle the infinite case are naturally able to handle the finite case. However, this case has an added real time restriction. The data must be processed quickly enough to keep pace with the incoming flow. If the algorithm is too slow the backlog will build up and eventually incoming data will be lost. The online case has the potential for concept drift. If the underlying concept shifts over time, the algorithm should be capable of adapting as necessary.

The method proposed in this paper is based on option trees [3]. The learning scheme is a variant of the ensemble method, where the model maintained is a single straightforward voting structure. The structure is such that it can be merged with linear complexity whereas merging regular decision trees is a multiplicative process [14]. To keep accuracy high, the models are induced via boosting. To restrict memory usage we introduce a method of pruning the model, rather than its members, that is simple and fast. The resulting algorithm can be adjusted according to speed and memory requirements. It has the potential to follow drifting concepts, although we do not deal with concept drift here.

This paper is organised in the following way; in the next section we outline our new approach for mining data streams using option trees. In Section 3 we present the experimental results obtained from our approach. Section 4 details related work and Section 5 contains concluding remarks.

2 Description of the Algorithm

Option trees [3] are a generalization of the standard decision tree structure. Multiple options can be explored at a conditional point, rather than the single path restriction of conventional trees. Figure 1 presents an example option tree. To classify an example where *age* = *unknown*, *sex* = *male*, *height* = 175, *weight* = 100 and *eyes* = *brown*, one would add the values $0.231 + 0.948 - 0.965 - 0.296$ to reach the sum -0.082. The sum is negative so the negative class would be assigned to this case.

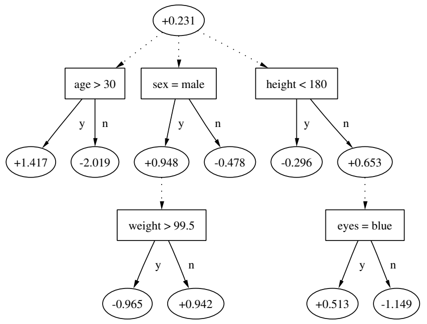

Fig. 1. An example option tree

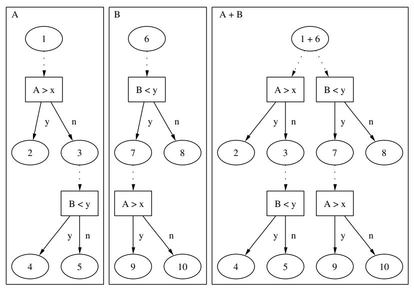

Fig. 2. Merging option trees

The problem of inducing an option tree from training data can be approached in many ways. In this paper we use the boosting algorithm introduced by Freund and Mason [7]. They defined so-called alternating decision trees (ADTrees), option trees generated via boosting.

2.1 Merging

The ability to merge several models into a single equivalent model is a desirable property when dealing with ensembles. A single universal model is easier to maintain and interpret than several disjoint ones. Things can be further complicated if each member is individually weighted. The smaller the merged model the better it will be for this purpose.

As Quinlan discovered when trying to merge standard decision trees, their combination is multiplicative [14]. Option trees, however, merge additively in the worst case. Depending on how much structure is shared, further savings can be made.

Figure 2 demonstrates the result of merging option trees A and B. The merging process starts at the root of the trees and works its way down, looking for any common tests at the same level to merge into one. If any tests match, it will merge the underlying prediction values by adding them together, and then merge the subsequent sub-trees. If there is no match, there is no choice but to add another option branch to the tree containing an unshared sub-tree.

With a limited range of nominal labels available one can see the savings to be made when common tests involving nominal attributes are merged. The merging of numeric tests is more troublesome. While there is certainly overlap between numeric ranges, there appears no obvious way of combining these tests without discarding information. The rule followed by the merging algorithms used in this paper is to only merge numeric tests if the split point is identical. As is often the case with inducing trees from different sub-samples of the data, this can result in multiple split points that differ ever so slightly. Sophisticated merging of numeric attributes, however, is not addressed in this paper.

In Figure 2 the only nodes that can be merged are the root nodes. Examination of the merging process shows that the order of tests in the trees prevents any further compression of the merged model. Trees A and B share common tests, but because they are evaluated in different order we fail to merge them. Further savings can be made by converting to an order-independent representation.

2.2 Flat representation

Each node of a decision tree is considered in the order encountered while traversing down the tree. A node is not considered unless its parents have been evaluated first. Maintaining this order when merging trees causes exponential growth. Consequently we seek an order-independent representation. The graph we construct is made up of two connected layers. The top layer consists of conditions; the bottom consists of prediction weights. The structure is no longer a tree as a prediction node can have multiple parents or no parents at all.

Figure 3 illustrates the result of merging example trees A and B in flattened form. Converting an option tree into this form is simple—we add all of the previously unseen condition nodes found in the tree to the top layer, and all of the prediction nodes to the bottom layer. We link the prediction nodes to each of the ancestor parent conditions present in the tree. For example, node 5 depends on test $A > x = n$ and $B < y = n$ in the tree, so we link node 5 to test $A > x$ and $B < y$ in the flat graph, noting the condition (=y or =n) of the connection.

There is a provision added to the graph construction to make it as compact as possible—the set of tests linked to a prediction node should be unique. In the case where adding a prediction node with the same test conditions as an existing node in the graph, the prediction values are added together rather than adding a new node to the graph. The ability to do this is the key to being able to merge the graphs more efficiently than the trees—the fact that nodes 10 and 4 share identical preconditions is detected despite the fact they are found at different depths in the original trees.

When represented this way, the structure can efficiently be merged with option trees or their flattened equivalents. Performing classification with this structure is a matter of summing all of the prediction nodes whose parent tests are satisfied. The final prediction sum derived from this is identical to the sum that would have been obtained by the option trees making up the model.

2.3 Rule representation

Visually, the two-layer graph representation can be hard to interpret. Typically there are many more prediction nodes than test nodes, and the links between them form a dense and complex web. To make the model easier for users to follow, it can instead be represented as a set of voting rules.

Figure 4 shows Figure 3 transformed into a set of voting instructions. Each prediction node becomes a rule—the prediction value is listed, along with the conditions required for the value to apply. To make a prediction, the user adds up values of the rules that hold true.

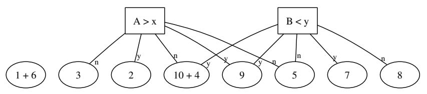

Fig. 3. Flattened graph of A + B. Where there is more than one edge leading to a predictor node, all of the tests must be satisfied for the prediction value to apply

Start with (1+6).
Add (2) if $A > x$.
Add (3) if $A <= x$.
Add (10+4) if $A <= x$ and $B < y$.
Add (9) if $A > x$ and $B < y$.
Add (5) if $A <= x$ and $B >= y$.
Add (7) if $B < y$.
Add (8) if $B >= y$.
If $sum < 0$ predict negative class,
if $sum > 0$ predict positive class,
if $sum = 0$ prediction unknown.

Fig. 4. Voting rules derived from figure 3. The voting weights correspond to the labels in figure 3—normally these would be real-valued numbers

The naïve approach to generating predictions by consulting every rule can be costly as the model grows. We overcome this problem by converting the model back into an option tree for prediction. This is achieved by taking the most frequent test as a root and partitioning into three sets, those for which the test holds, those for which it does not hold and those that do not test that particular attribute. The sets are recursively divided down respective paths of the tree.

Conceptually, it helps to rank the rules by their absolute prediction value. Those rules with larger weights have potentially more influence on the outcome than smaller weighted rules. This observation suggests a simple pruning technique.

2.4 Pruning

In general, one would expect a rule with a weight close to zero to have little influence on a model's classifications. The only cases where these rules would make a difference is when the decision is borderline—in which case a small value may be enough to push over the classification boundary, or when many of them combine to form a large overall difference.

It is in the fine details that a model is able to describe a complex relation, and removing some of these details could damage the models performance. However given a choice between sacrificing the large details versus the small ones the logical choice is to hold on to the seemingly most important ones. Here we adopt a simple philosophy: when running out of space, the smallest weighted rules are the first to go.

Figure 5 justifies this choice. It shows an example of the effect that pruning has on accuracy when three different removal strategies are used. The strategy of removing the largest weighted rules first demonstrates a sharp performance decay, whereas removing the smallest weights first has the least impact on accuracy. In fact, there are significant stretches where removing the smallest nodes has little impact on prediction error—it is apparent that the 600 or so smallest weights can be removed with virtually no performance loss, as evidenced by the horizontal stretch at the left hand side of the graph.

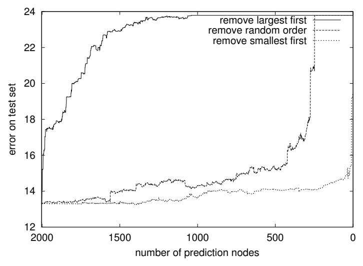

Fig. 5. Effect removal order has on test set accuracy when pruning back a full model trained on the adult dataset

Also present in the graph is an example of removing nodes in random order. The shape of this curve varies according to the randomization. On average the accuracy degradation of random removal falls somewhere between the two extremes of largest-first and smallest-first.

When merging a new model, the operation can be broken down into the individual insertions of prediction values. To keep the model memory usage in check, an upper limit on the number of prediction nodes is set. Once this limit is hit, and the insertion of a new prediction value requires the creation of a new node, the prediction node with the smallest magnitude is discarded to make space for the new node. In this way, an upper bound on the memory requirements for the model is maintained.

2.5 Chunking

```
1. Initialize global model G
2. Initialize chunk buffer to chunk size
3. While incoming data is available
    3.1. Add next instance to chunk buffer
    3.2. If chunk buffer is full
        3.2.1. Learn a model M from chunk buffer
        3.2.2. Merge model M with global model G
        3.2.3. Clear chunk buffer
    End if
End while
```

Fig. 6. Chunking algorithm

Figure 6 outlines the algorithm for learning a model from a data stream. The first two steps prepare for the incoming data stream. Step 3.2.1 employs the alternating decision tree learning algorithm to induce an option tree from the most recent chunk of data. The next step carries out a merging operation to update the overall model. It is during this step that the global model may be pruned to ensure it does not exceed the maximum allowable size. At any stage should we wish to perform a classification on an unknown data instance, we can use the global model resulting from the chunking algorithm.

3 Experiments

Our experiments aim to observe the influence the chunk size, the number of boosting iterations per chunk, and the maximum number of prediction values allowed in the model have on classification accuracy, learning speed, and memory consumption against a finite data stream.

3.1 Datasets and Methodology

To overcome the lack of real data of sufficient size, authors often generate synthetic data sets. Typically the synthetic data is used to present scalability results,

Table 1. Datasets used for the experiments

Dataset	Train	Test	Numeric	Nominal
anonymous	32711	5000	0	293
adult	38842	10000	6	8
census-income	249285	50000	8	33
covertype	395141	100000	10	44

prediction accuracy results are rarely reported. We collected the few suitable datasets we could find, mostly from UCI [2], and synthetically generated some more. To benchmark our ensemble method we use the WEKA [1] implementations of C4.5 and ADTree to build a model over the entire data. The datasets are given in Table 1. The accuracy results are based on an independent test set, the size of which is listed in the table. Note that the train/test splits are not the original ones supplied with the datasets—to ensure an equally distributed sample we randomized the data before splitting it up.

3.2 Results

The speed of model growth is dependent on the chunk size and the number of boosting iterations. Decreasing the chunk size generates sub-models more frequently, and increasing boosting iterations creates larger sub-models, both of which cause the overall model to grow at a faster rate.

Growth curves on datasets are close to linear in the number of training examples. One point of difference is that the growth of the number of tests varies on the commonality of tests. Greater numbers of nominal attributes in the data improve the chance for merging, whereas for numerical attributes splitpoints are rarely replicated thus reducing the opportunities for merging [2]. The results tend to suggest that although merging of common nodes takes place, it is not very substantial, hence the almost worst-case linear behaviour.

Assuming an infinite supply of data, it would be ideal to use chunk sizes as large as possible. This would ensure that training samples are as representative as possible of the underlying distribution. The two constraints on chunk size are memory and complexity. The chunk size must be small enough to fit into memory along with the memory required to train the models. Larger chunk sizes slow the algorithm's ability to process data.

In theory, one would assume that in determining chunk size there is a minima—where below this point insufficient data is supplied to the learning algorithm to build models representative of the overall problem, and above which fewer and fewer gains are to be made. This would vary depending on the dataset. Automatic determination of the minima for a given dataset is a topic for further

[1] Waikato Environment for Knowledge Analysis (http://www.cs.waikato.ac.nz/ml)

[2] With our policy of requiring identical splitpoints for a merge, there are a theoretically infinite number of tests available for numeric attributes.

research. One possibility is to race several candidates as in [6]. In this paper we only investigate the effect of a small number of different fixed chunk sizes.

Figures 7,10,13 and 16 show the effect on prediction accuracy when the chunk size is varied on four datasets. In all cases 50 boosting iterations were used per chunk. As expected, smaller chunk sizes lead to more erratic accuracy, and larger chunks have smoother curves. This effect is partly exaggerated due to the larger-chunk graphs being plotted with larger horizontal steps. Choosing a chunk size too low can severely hurt learning performance—the smallest chunk sizes are the worst performers, except in one case. On covertype, the largest chunk size reaches the best performance early, but later a strange turn-around occurs.

We only explore this range of chunk sizes because we only have so much data available, ramping the chunk size too high means the data is quickly exhausted without providing an appreciation of trends over time. By fixing the chunk size to 4000, we can see the effect of changing the amount of boosting. Figures 8,11,14 and 17 show the effect on prediction accuracy when the number of boosting iterations ranges from 20 to 160.

In general, the more boosting iterations the more accurate the sub-models, which leads to better committee performance. Similar to the chunk size, the committee is starved if the parameter is set too low. On these datasets, boosting 80 times does a reasonable job, with 160 providing little gain. This is not to say that increasing the boosting iterations further will not improve things more, just that the gains are diminishing. The boosting parameter is flexible and can be adjusted according to requirements. Boosting 160 times becomes quite computationally demanding so we stopped at this point.

Fixing the chunk size to 4000 and the boosting iterations to 80, we investigate the effect of pruning on the accuracy of the model. In figures 9,12,15 and 18 we restrict the number of prediction nodes allowed in the model, using the pruning technique outlined in Section 2.4, and see the effect it has on accuracy.

It is clear from these graphs that setting the memory restrictions too low can have a serious impact on the learning capacity of the algorithm. At the lowest memory usage, the error tends to drift upwards over time. It must be considered that the kind of memory restrictions we are enforcing are very severe—in the order of kilobytes for storing the models, where modern computing resources would effortlessly allow many megabytes for model storage.

Table 2 compares the accuracy of our technique against other methods. The first three columns show the error obtained by training C4.5, boosted C4.5 and ADTree (both boosted 80 times) respectively on the entire training set. The fourth and fifth columns (C4.5C, BC4.5C) shows the accuracy of a committee of C4.5 and boosted C4.5 trees generated under similar conditions to our technique (chunk size of 4000). The OTC (Option Tree Committee) column shows the error obtained using our chunking algorithm with no pruning, chunk size 4000 and 80 boosting iterations. Clearly it is not always possible to apply boosting to these datasets. The DNF results refer to out of memory exceptions at 1GB and cover an inability to load an entire dataset and an inability to cope with an ever increasing number of large models.

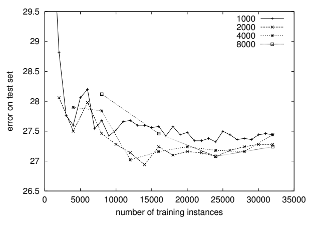

Fig. 7. anonymous/chunk

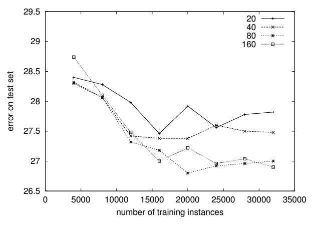

Fig. 8. anonymous/boost

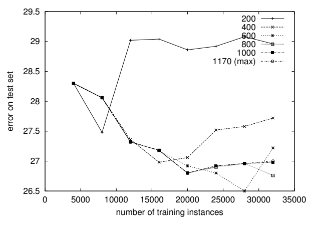

Fig. 9. anonymous/size

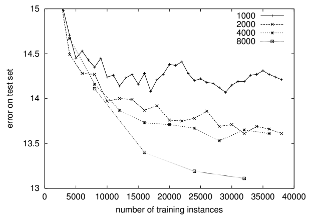

Fig. 10. adult/chunk

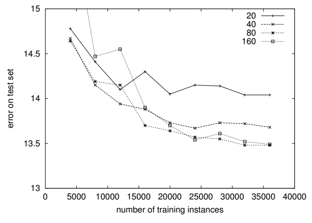

Fig. 11. adult/boost

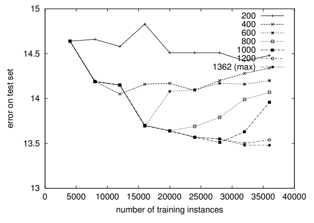

Fig. 12. adult/size

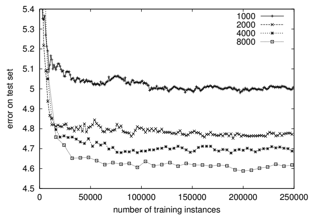

Fig. 13. census/chunk

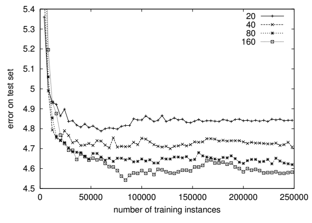

Fig. 14. census/boost

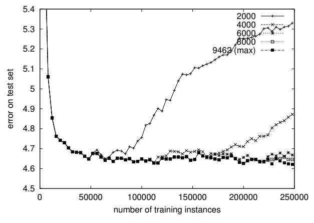

Fig. 15. census/size

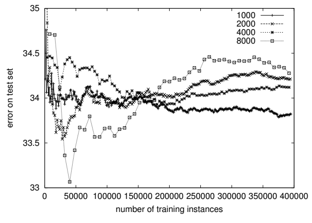

Fig. 16. covertype/chunk

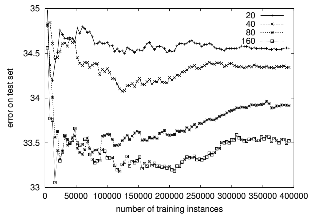

Fig. 17. covertype/boost

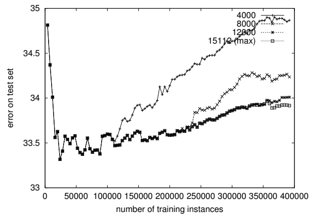

Fig. 18. covertype/size

Table 3 shows the corresponding sizes of the models. The size is measured by the number of rules contained in the model, that is, the total number of leaves in C4.5, and the number of prediction nodes in the option tree methods. The size of the ADTree models is constant due to the fixed number of boosting iterations. The size of the unpruned OTC models are generally larger than the others. The boosted C4.5 rule sizes were not included but are up to 80 times larger than their non boosted counterparts.

These results are interesting. First, OTC matches ADTree accuracy on anonymous and covertype. It betters C4.5 on the other two datasets. According to McNemar's test [4] the differences between C4.5 and OTC on adult and census are not significant, whereas the differences between C4.5 and ADTree are. C4.5 is clearly superior to the others on the anonymous and covertype data. The boosting results, although patchy, do indicate that chunking can be more beneficial than using the entire dataset, which supports the argument for an OTC style method.

We tested the scalability of our method with synthetic data, the results we omit for lack of space. The conclusion we could draw from the analysis was that even when faced with millions of instances OTC scales linearly in run-time and space requirements. To give a concrete example, on the particular test hardware (an Athlon XP 1700+) it was possible to learn from over 300 instances per second, and classify 1000 instances per second after training 1 million instances with no model size restriction. The model sizes reached several megabytes at the most.

Table 2. Error on test set

Dataset	C4.5	BC4.5	C4.5C	BC4.5C	ADTree	OTC
anonymous	24.60%	DNF	26.86%	24.94%	27.20%	27.00%
adult	13.67%	15.11%	14.12%	13.85%	12.92%	13.48%
census-income	4.69%	4.72%	5.44%	DNF	4.46%	4.62%
covertype	29.84%	DNF	33.10%	DNF	34.57%	33.54%

Table 3. Number of rules in model

Dataset	C4.5	C4.5C	ADTree	OTC
anonymous	611	662	201	1170
adult	495	1186	201	1362
census-income	2480	1675	201	9462
covertype	8297	19946	201	15112

4 Related Work

Very large datasets have inspired several interesting systems, mainly based on decision trees. Mehta et al. made one of the first attempts to scale decision trees to large datasets with SLIQ [13]. The idea was improved in SPRINT [15]. CLOUDS [1] introduced methods of approximating numeric split points to improve efficiency. RainForest [9] is a framework for further reducing the computation required, that generalizes to all of these approaches. BOAT [8] reduces the number of passes required by building from samples, and correcting as necessary. All of these methods require multiple scans of the data, and are thus not suitable for data streams.

Domingos and Hulten introduce VFDT [5], a method for building decision trees from high speed data streams. They use Hoeffding bounds [10] to guarantee performance. The approach is improved by Jin and Agrawal in [12]. In [11] Domingos and Hulten go on to claim that any method based on discrete search can be adapted by their technique.

The approaches most closely related to this paper use ensembles. Street and Kim propose SEA [16], in which models are bagged over chunks of a data stream and weighted. Frank et al. [6] boost subsequent models and prune those that fail to improve performance. Chunk sizes are raced to determine optimal size. None of these techniques however maintain a single classification model.

5 Conclusions

We have presented an algorithm for mining data streams using option trees. The method has three correlated parameters, chunk size, number of boosting iterations and model size. Experiments have been performed to observe the influence these have on classification accuracy, learning times, and memory consumption.

These initial experiments have not determined optimal parameter settings, indeed these may well be dataset dependent, but generally large chunk sizes with high numbers of boosting iterations will give rise to good classification performance. Also, the pruning of small prediction nodes can maintain a small model without too much loss in predictive accuracy. The maximum memory allocation investigated in this paper is under 4MB when processing five million instances, so there is much scope for much greater memory utilisation.

Training models using OTC is linear in the number of instances fulfilling the scalability requirements of the data stream model. Option trees present a realistic solution to the problem of mining data streams. They can be boosted to obtain good performance, merged to maintain a single transparent model and pruned to fit in available memory. Results from initial experiments are very encouraging when compared to traditional methods that can view all instances at once.

There are many avenues for future work to improve the method presented in this paper. Possible directions to explore include merging numeric attribute ranges to optimise tests, improving classification time, inducing option trees via other means, tracking concept drift and dealing with multi-class problems.

References

1. Khaled Alsabti, Sanjay Ranka, and Vineet Singh. CLOUDS: A decision tree classifier for large datasets. In *Knowledge Discovery and Data Mining*, pages 2–8, 1998.
2. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
3. W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall, London, 1993.
4. T. Dietterich. Statistical tests for comparing supervised classification learning algorithms, 1996. Technical Report, Department of Computer Science, Oregon State University, Corvallis, OR.
5. P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
6. Eibe Frank, Geoffrey Holmes, Richard Kirkby, and Mark Hall. Racing committees for large datasets. In *International Conference on Discovery Science*, 2002.
7. Yoav Freund and Llew Mason. The alternating decision tree learning algorithm,. In *Proc. 16th International Conf. on Machine Learning*, pages 124–133. Morgan Kaufmann, San Francisco, CA, 1999.
8. Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. BOAT — optimistic decision tree construction. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, pages 169–180, 1999.
9. Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest - a framework for fast decision tree construction of large datasets. *Data Mining and Knowledge Discovery*, 4(2/3):127–162, 2000.
10. W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
11. Geoff Hulten and Pedro Domingos. Mining complex models from arbtrarily large databases in constant time. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 2002)*, pages 525–531, 2002.
12. Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.
13. Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32, 1996.
14. J. Quinlan. Miniboosting decision trees, 1999. Submitted to JAIR (available at http://www.cse.unsw.edu.au/∼quinlan/miniboost.ps).
15. John C. Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 544–555. Morgan Kaufmann, 1996.
16. W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2001.