Working Paper Series
ISSN 1170-487X

Mining Data Streams Using Option Trees

Geoffrey Holmes, Bernhard Pfahringer and Richard Kirkby

Working Paper: 08/03
September 2003

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Mining Data Streams Using Option Trees

Geoffrey Holmes, Richard Kirkby, 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.

Keywords: *classification, option trees, ensemble methods, data streams*

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*).

A finite stream flows from a large but finite data source. By treating the problem as a data stream, the one-pass requirement means an algorithm will scale linearly with the size of the data. This makes it faster at mining large datasets than multi-pass approaches. The updatability requirement ensures that new training data can be incorporated efficiently as it becomes available.

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.

Some traditional methods can be incrementally updated, for example, Naïve Bayes [13] and Utgoff's ID5R [21] (improved in [22]). Better predictive accuracy than Naïve Bayes can be obtained with more sophisticated techniques, perhaps the most popular being the decision tree [4, 17]. ID5R is an incremental decision tree algorithm, but it is not suitable for data streams as it needs to remember all of the training data it observes. There are several attempts to scale decision trees to large datasets [14, 19, 10, 11, 2], a few of which already meet the single pass criteria [7, 12]. Because the view of data is limited and incremental, one of the biggest problems these algorithms face is determining the optimal split point values for numeric attributes.

It is desirable to couple decision trees with boosting [18] as it is generally accepted that boosting can improve the predictive performance of an algorithm. Boosting entire models over a finite data stream, however, is prohibitively expensive, or impossible in the infinite case.

Another approach to tackling data streams is to use ensembles. This involves collecting a group of classifiers, each trained on a small portion of the data, and collating

their votes to classify new examples. Examples of such an approach include [23, 20]. A nice feature of this technique is that existing learning schemes can be used at the base level, leaving the problem of handling the data stream to the meta-level algorithm. Various methods of weighting and pruning the ensemble can be explored to improve predictive performance. Pruning is necessary to ensure that memory restrictions are obeyed.

A drawback of the general ensemble technique is that the model will contain multiple sub-models. Each of these models may be understandable on their own, but as a collective their reasoning is less transparent.

The method proposed in this paper is based on option trees [5]. 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 [16]. To keep accuracy high, the models are induced via boosting, and the decision tree problem with numeric splits is overcome in a simple way. 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

This section begins by describing option trees, the basis for our technique. Next is a discussion about merging option trees. We show how they can be transformed to an order independent form, and as a set of rules for transparency. Following this is a description of our pruning technique, and the overall algorithm for learning from a data stream using chunks of data.

2.1 Option trees. Popular decision trees, of the like produced by algorithms C4.5 [17] and CART [4], consist of a structure of connected nodes, originating from a root node. Each node may either contain a condition, in which case it splits the data among its children, or a leaf (childless) node containing a prediction. To perform classification on an unlabelled data instance, one traverses the single unique path [1] down the tree determined by the condition nodes until a leaf is encountered—the instance is assigned the prediction found in the leaf.

Option trees [5] are a generalization of the standard decision tree structure. They add the possibility of exploring several options at a conditional point, rather than following a single path. This means that performing classification with an option tree is less straightforward—it involves traversing all paths the data instance satisfies, and aggregating prediction information along those paths to reach a decision.

To facilitate the extra information required in the tree, prediction nodes are introduced. This paper deals only with two-class problems, in which the prediction nodes contain a single real valued weight—the more negative the value the more it tends towards one class, the more positive the more it tends towards the opposite class.

The final classification of the tree is derived by summing the prediction values encountered along all of the paths satisfied by the instance in question. If the sum is negative, the negative class is predicted, and likewise for the positive class. Additionally, the magnitude of the sum can give extra information about the confidence of the prediction, sometimes referred to as the margin.

It is clear that the model space available in the option tree representation exceeds that of a standard tree. In fact, an option tree can represent a voted collection of standard 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.

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 [9]. They defined so-called alternating decision trees (ADTrees), option trees generated via boosting. Boosting enables the generation of highly accurate models.

Alternating decision trees are generated by considering, at each iteration, the addition of all possible tests under all prediction nodes present in the tree. The test which minimizes an impurity function is selected and added to the tree. The weight of each training instance is altered to reflect the result of adding the

[1] Actually, C4.5 will explore multiple branches when dealing with missing values

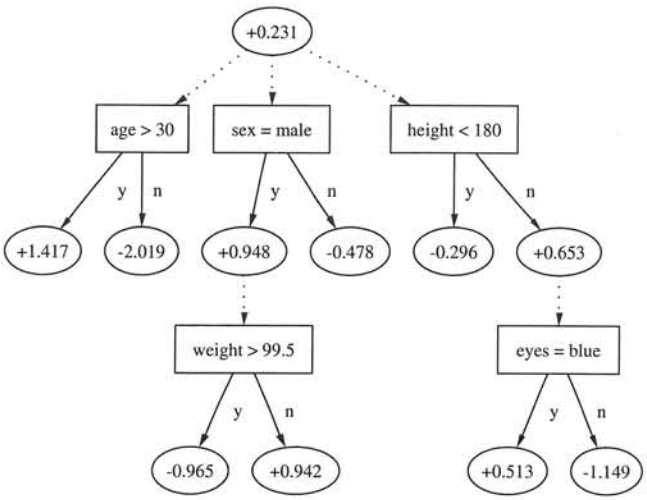

Figure 1: An example option tree.

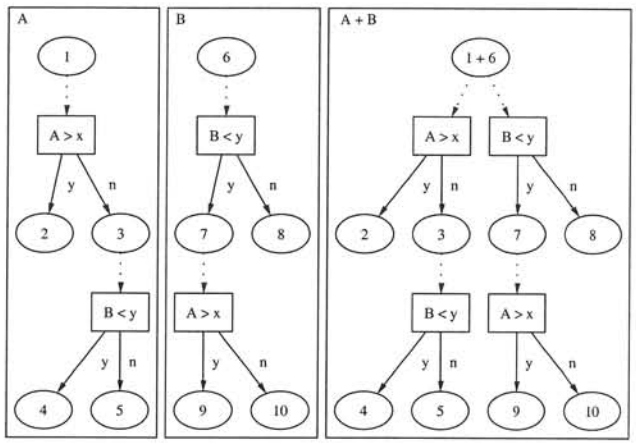

Figure 2: Merging option trees.

new test—those that are classified correctly decrease in weight, while those that are incorrectly classified have their weight increased. This procedure is at the core of boosting—the subsequent model concentrates on the mistakes of the previous. The cycle is continued until a tree of the desired size is constructed. There is no established stopping criterion for the algorithm other than using cross-validation to determine the best point. In our experiments we try fixing the number of boosting iterations at various levels to observe the effects.

2.2 Merging. The ability to merge several models into a single equivalent model is a desirable property when dealing with ensembles. The reasoning is that 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, the combination is multiplicative [16]. This rules them out as candidates for building a universal model. Option trees 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 throwing information away. 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-dependent nature of the trees prevents even 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.

2.3 Flat representation. Decision trees, by their nature, are order-dependent. Each node is considered in the order encountered while traversing down the tree. A node is not considered unless its parents have been evaluated first.

Here we present an order-independent graph representation, which 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.

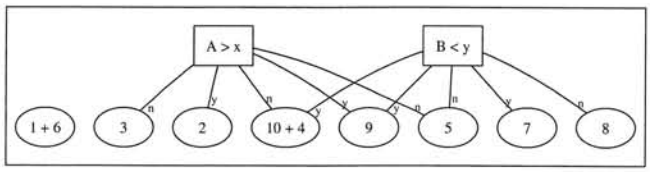

Figure 3: Flattened graph of A + B. Where there is more than one edge leading to a predictor node, all of the tests must be satisfied for the prediction value to apply.

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.

2.4 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.

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.

Figure 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. There are some ways to speed up classification that are not explored in this paper. Firstly, the problem of testing and summing over a large number of prediction nodes is one that is easily parallelized—so a solution could be to share the task among multiple processors. Secondly, the process can be optimized. One method would be to start with the heaviest weights and stop as soon as it is determined that the smaller weights cannot change the result. Thirdly, we could find the most frequent test, say "A == v", then partition into three sets. Rules for which the test (A==v) is true (and false), and rules that do not test A == v. Depending on the value of A, we only need examine two of these sets of rules.

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. It is this observation that suggests a simple pruning technique.

2.5 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. By this reasoning,

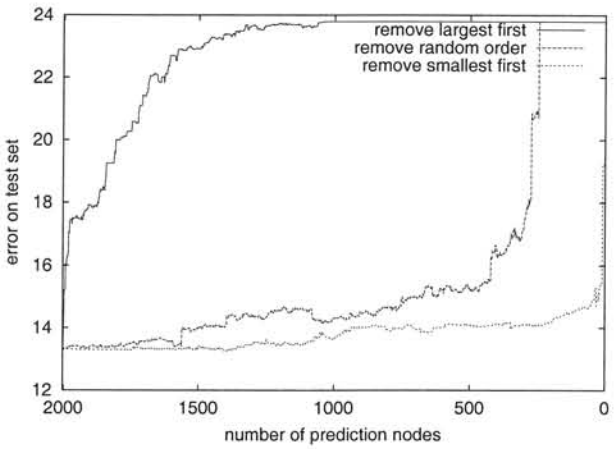

Figure 5: Effect removal order has on test set accuracy when pruning back a full model trained on the adult dataset.

the pruning method in this paper follows this simple philosophy: when running out of space, the smallest weighted rules are the first to go.

Figure 5 justifies this concept. 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.

Also present in the graph is an example of removing nodes in random order. The shape of this curve varies according to the randomization. It is sensible to assume that on average the accuracy degredation of random removal will fall somewhere between the two extremes of largest-first and smallest-first, as is demonstrated.

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, while the most influential parts of the model are preserved.

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

Figure 6: Chunking algorithm.

2.6 Chunking. 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 is the loop that processes the 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, 3.2.2, 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

The learning algorithm presented in this paper has three parameters that can be controlled—the chunk size, the number of boosting iterations per chunk, and the maximum number of prediction values allowed in the model. The chunk size and boosting iterations have an affect on the run-time complexity of the algorithm. Under the real-time online constraint, one would set these parameters to ensure that learning time can keep pace with the data flow. The third parameter puts a cap on the memory usage of a model, at the potential expense of accuracy.

Our experiments aim to observe the influence these parameters have on the outcomes of most interest—classification accuracy, learning speed, and memory consumption against a finite data stream.

3.1 Datasets and Methodology. The most commonly used datasets for evaluating scalable tree algorithms appear to be those of STATLOG [15]. Unfortunately, these datasets are relatively small, the majority of which can hardly be considered large-scale, the smallest containing only 690 examples, the largest 57,000 examples.

Dataset	Train	Test	Numeric	Nominal
anonymous	32711	5000	0	293
adult	38842	10000	6	8
census-income	249285	50000	8	33
synthF7	5000000	1000000	7	2

Table 1: Datasets used for the experiments.

The problem of finding benchmark datasets with which to compare our results with is further compounded since we are restricted to two class problems. This rules out all of the STATLOG datasets bar the smallest two. We could transform some of the multi-class datasets to two class problems, but the results would not be comparable to anything published.

Often, to overcome the lack of data, authors will synthetically generate data sets. The most common method of synthesizing data in the literature appears to be the tool described in [1]. Typically the synthetic data is used to present scalability results, and rarely are prediction accuracy results reported.

To overcome these deficiencies we collected the few suitable datasets we could find, mostly from UCI [3], and synthetically generated some more. To benchmark our ensemble method versus other methods, we use the WEKA [2] 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.

The synthF7 dataset was generated using the technique in [1] using predicate function 7 with default parameters.

In order to measure the running time of the algorithms we use the user time returned by the Unix time command. The experiments were carried out on an AMD Athlon XP 1700+ with 512MB RAM.

3.2 Results. When examining model growth a common pattern emerges. Figure 7 illustrates growth on the adult data, which is a typical example of behaviour. Model growth is very close to linear in the number of training examples processed. It is slightly sub-linear thanks to the merging of nodes that takes place, as is most evident when looking at the number of test nodes—this is where most replication occurs.

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

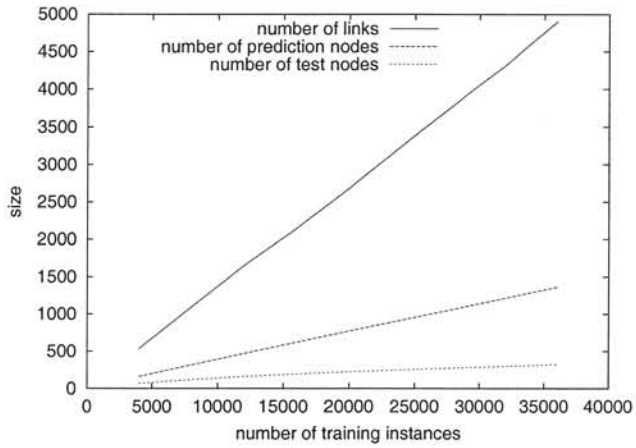

Figure 7: Model growth on the adult dataset (chunk size 4000, 80 boosting iterations).

The growth curves on other datasets look very similar, the one difference perhaps being 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 [3].

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 the samples trained on 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.

[2] The Waikato Environment for Knowedge Analysis, available at http://www.cs.wakato.ac.nz/ml

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

Larger chunk sizes also 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 research. One possibility is to race several candidates as in [8]. In this paper we only investigate the effect on accuracy that a small number of different fixed chunk sizes have.

Figures 8 through 10 show the effect on prediction accuracy when the chunk size is varied on three 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. It is clear that choosing a chunk size too low can severly hurt learning performance—the smallest chunk sizes are consistently the worst performers.

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 11 through 13 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 14 through 16 we restrict the number of prediction nodes allowed in the model, using the pruning technique outlined in Section 2.5, 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

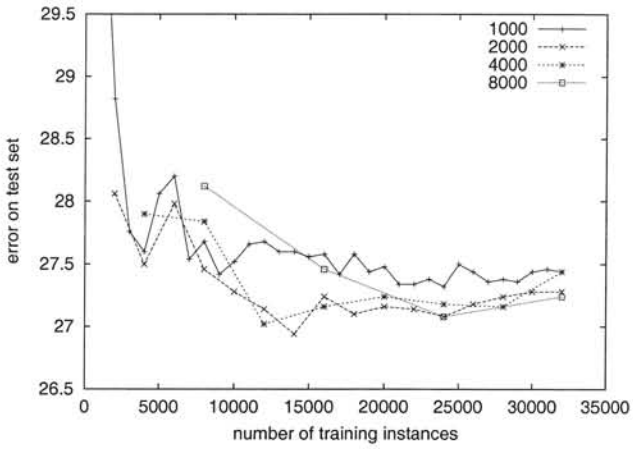

Figure 8: Varying chunk size on anonymous.

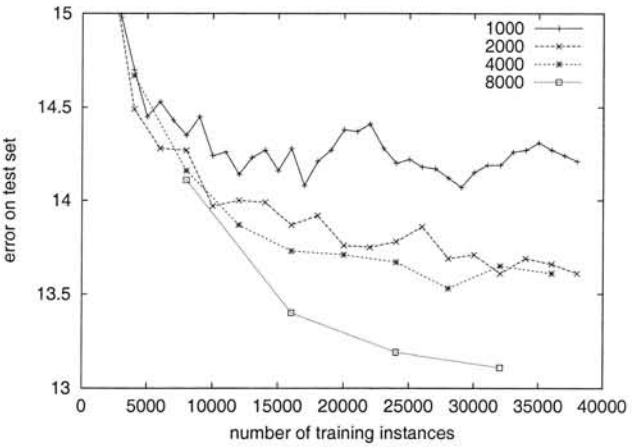

Figure 9: Varying chunk size on adult.

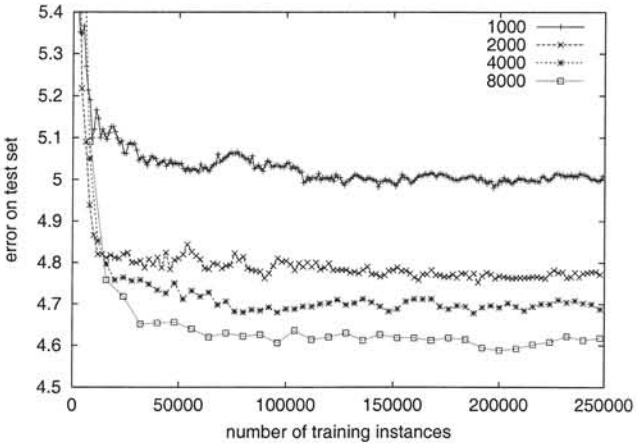

Figure 10: Varying chunk size on census-income.

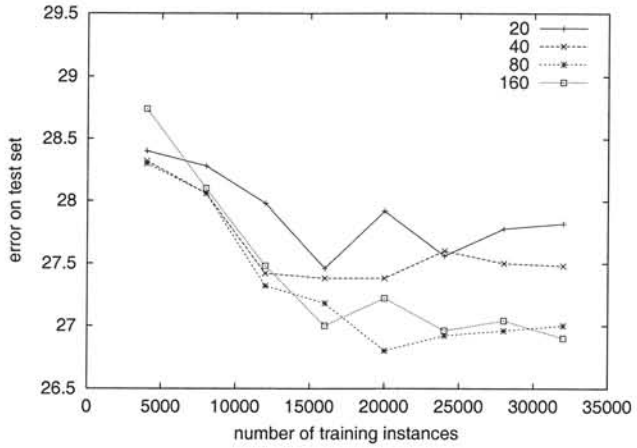

Figure 11: Varying boosting iterations on anonymous.

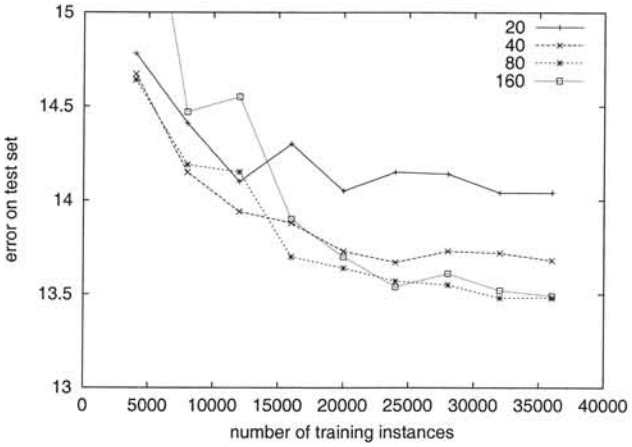

Figure 12: Varying boosting iterations on adult.

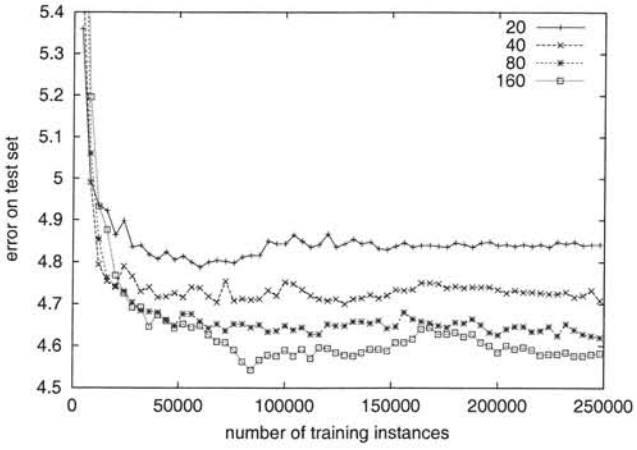

Figure 13: Varying boosting iterations on census-income.

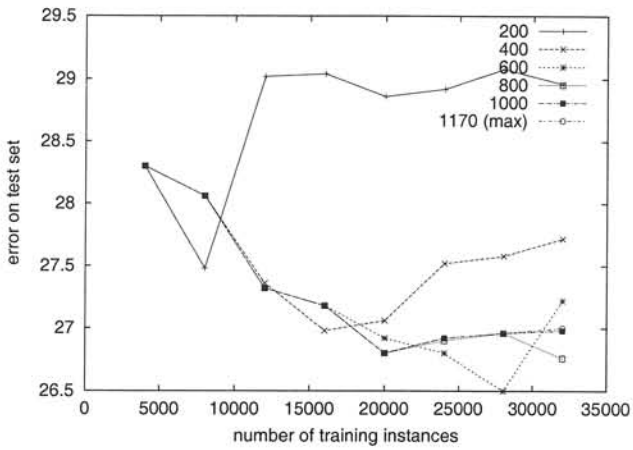

Figure 14: Varying maximum model size on anonymous.

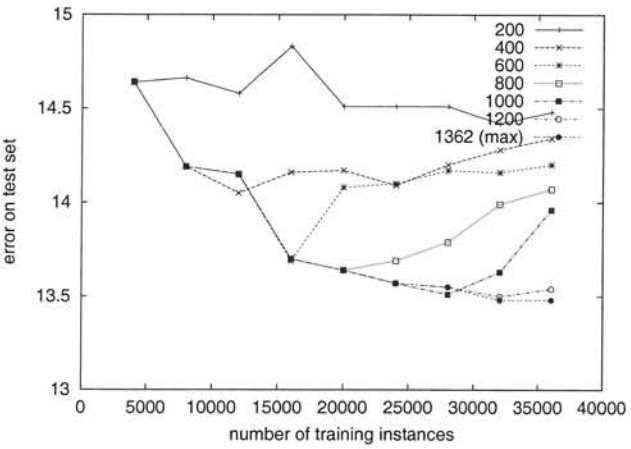

Figure 15: Varying maximum model size on adult.

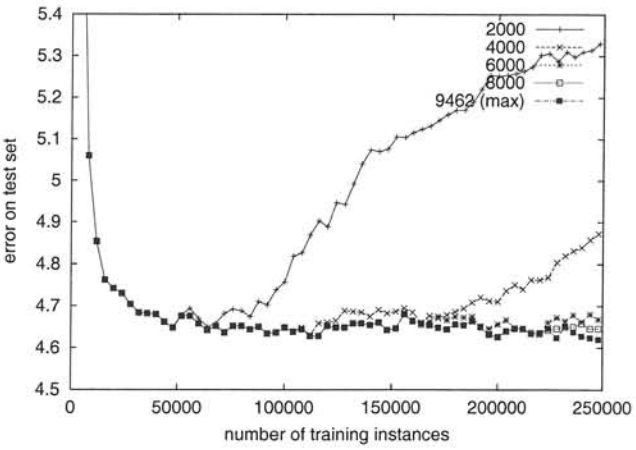

Figure 16: Varying maximum model size on census-income.

Dataset	C4.5	ADTree	OTC
anonymous	24.60%	27.42%	27.00%
adult	13.67%	12.83%	13.48%
census-income	4.69%	4.48%	4.62%

Table 2: Error on test set.

Dataset	C4.5	ADTree	OTC
anonymous	611	201	1170
adult	495	201	1362
census-income	2480	201	9462

Table 3: Number of rules in model.

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 two columns show the error obtained by training C4.5 and ADTree (boosted 100 times) respectively on the entire training set. The OTC (Option Tree Committee) column shows the error obtained using our chunking algorithm with no pruning, chunk size 4000 and 80 boosting iterations.

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 number of leaves in C4.5, and the number of prediction nodes in the other two methods. The size of the ADTree models is constant due to the fixed number of boosting iterations. The size of the unpruned OTC models is significantly larger than the other two.

Table 2 contains some interesting results. First, OTC matches ADTree on anonymous and C4.5 on the other two datasets. According to McNemar's test [6] the differences between C4.5 and OTC on the last two datasets are not significant, whereas the differences between C4.5 and ADTree are. C4.5 is clearly superior to the others on the anonymous data.

To investigate the scalability of our algorithm we use synthetically generated data. We used a chunk size of 5000, boosted 20 times, and no pruning. Figure 17 shows the training and testing times when scaling from 1 to 5 million training instances. The testing time is the time it took to classify 1 million instances after training to that point.

The first obvious point is that the testing time is worse than the training time. This is using a simple approach to generate classifications, Section 2.4 discussed how classification times may be improved. The important fact is that both training and testing

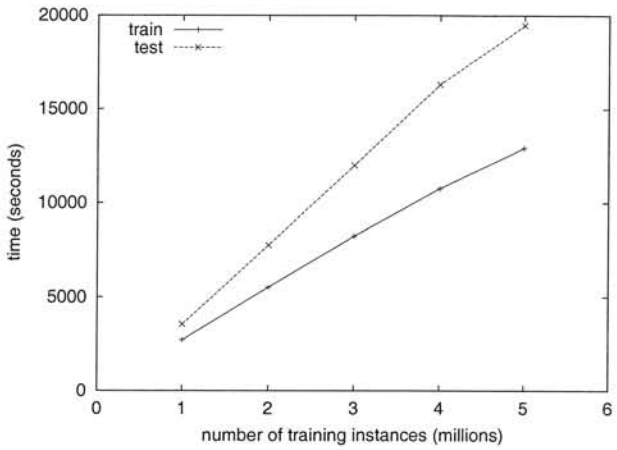

Figure 17: Run times on synthF7.

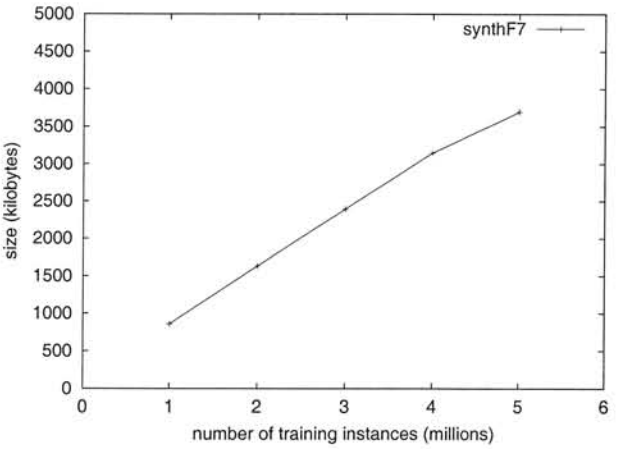

Figure 18: Sizes of the models output as Java serialized objects.

times scale linearly with the size of the data.

Figure 18 shows the sizes of the unpruned models when represented as Java serialized objects, which is the standard way of achieving persistence in Java. Note that we did not try to optimize storage efficiency in any way. Once again the linear nature of the algorithm is exhibited.

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 [14]. The idea was improved in SPRINT [19]. CLOUDS [2] introduced methods of approximating numeric split points to improve efficiency. RainForest [11] is a framework for further reducing the computation required, that generalizes to all of these

approaches. BOAT [10] 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 [7], a method for building decision trees from high speed data streams. They use Hoeffding bounds to guarantee performance. The approach is improved by Jin and Agrawal in [12].

The approaches most closely related to this paper use ensembles. Street and Kim propose SEA [20], in which models are bagged over chunks of a data stream and weighted. Frank et al. [8] boost subsequent models and prune those that fail to improve performance. Chunk sizes are raced to determine optimal size. In [23] Wang et al. explore weighted ensemble classifiers on concept-drifting data streams. None of these techniques maintain a single classification model, but a committee of distinct sub-models.

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. Of some concern is the time taken to classify unknown instances which although linear has a higher gradient than the time taken to train models. Some ideas for improving this aspect of the algorithm are presented in Section 2.4, but as yet none have been implemented.

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] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: A performance perspective. In Nick Cercone and Mas Tsuchiya, editors, *Special Issue on Learning and Discovery in Knowledge-Based Databases*, number 5(6), pages 914–925. Institute of Electrical and Electronics Engineers, Washington, U.S.A., 1993.

[2] 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.

[3] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[5] W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall, London, 1993.

[6] T. Dietterich. Statistical tests for comparing supervised classification learning algorithms, 1996. Technical Report, Department of Computer Science, Oregon State University, Corvallis, OR.

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

[8] Eibe Frank, Geoffrey Holmes, Richard Kirkby, and Mark Hall. Racing committees for large datasets. In *International Conference on Discovery Science*, 2002.

[9] 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.

[10] 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.

[11] 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.

[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] Pat Langley, Wayne Iba, and Kevin Thompson. An

analysis of bayesian classifiers. In *National Conference on Artificial Intelligence*, pages 223–228, 1992.

[14] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32, 1996.

[15] D. Michie, D. J. Spiegelhalter, and C. Taylor. *Machine learning, Neural and Statistical Learning.* Ellis Horwood, 1994.

[16] J. Quinlan. Miniboosting decision trees, 1999. Submitted to JAIR (available at http://www.cse.unsw.edu.au/~quinlan/miniboost.ps).

[17] R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Francisco, 1993.

[18] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Computational Learing Theory*, pages 80–91, 1998.

[19] 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, 3–6 1996.

[20] 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.

[21] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[22] Paul E. Utgoff. An improved algorithm for incremental induction of decision trees. In *International Conference on Machine Learning*, pages 318–325, 1994.

[23] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.