Batch-Incremental Learning for Mining Data Streams

Geoffrey Holmes
Department of Computer Science
University of Waikato
Hamilton, New Zealand
geoff@cs.waikato.ac.nz

Richard Kirkby
Department of Computer Science
University of Waikato
Hamilton, New Zealand
rkirkby@cs.waikato.ac.nz

Bernhard Pfahringer
Department of Computer Science
University of Waikato
Hamilton, New Zealand
bernhard@cs.waikato.ac.nz

ABSTRACT

The data stream model for data mining places harsh restrictions on a learning algorithm. First, a model must be induced incrementally. Second, processing time for instances must keep up with their speed of arrival. Third, a model may only use a constant amount of memory, and must be ready for prediction at any point in time. We attempt to overcome these restrictions by presenting a data stream classification algorithm where the data is split into a stream of disjoint batches. Single batches of data can be processed one after the other by any standard non-incremental learning algorithm. Our approach uses ensembles of decision trees. These tree ensembles are iteratively merged into a single interpretable model of constant maximal size. Using benchmark datasets the algorithm is evaluated 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.

There are two general approaches to building models incrementally. Models can either be adapted on the basis of a single instance or multiple instances. Examples of instance incremental algorithms include Naïve Bayes [12] and Utgoff's ID5R [18] (improved in [19]). At this point it is helpful to make a further qualification to these approaches by judging them on their memory usage. ID5R belongs to the class of algorithms that use unbounded memory (with the requirement to store training data) whereas Naïve Bayes is an example of a bounded memory instance incremental method. For multiple instance or batch incremental methods any standard machine learning algorithm can be used but all methods will require unbounded memory, effectively a bound will have to be placed on the number of batches. The method outlined in this paper attempts batch incremental learning with bounded memory (on a possibly infinite number of batches).

The underlying learning algorithm used in this paper is a standard decision tree but the method differs significantly from other attempts to scale decision trees to large datasets [13, 16, 8, 9, 2]. Not all of these methods are single-pass. Some require multiple passes due to problems such as determining the optimal split point values for numeric attributes.

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 [20, 17]. 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 learning scheme presented in this paper 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 straightforward merging of regular decision trees is a multiplicative process [14]. 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. 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

Popular decision trees, of the kind produced by algorithms such as C4.5 [15] 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 value or the probability that an instance belongs to one of the classes as a distribution. This structure has set the benchmark for classification accuracy. Not only is the tree accurate but it is transparent, a highly desirable feature.

It is not surprising then that standard decision trees have become popular in the streaming context. However, as Quinlan discovered when trying to merge standard decision trees, the combination is multiplicative [14]. This would relegate them to the class of batch incremental algorithms that use unbounded memory. It is possible, however, to use an intermediate representation of the decision tree that permits merging as a linear operation, and maintains transparency.

This representation and operation are described in the next section. We show how decision trees can be transformed to an order independent form, and to 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. All these subsections deal with the need to keep memory consumption bounded.

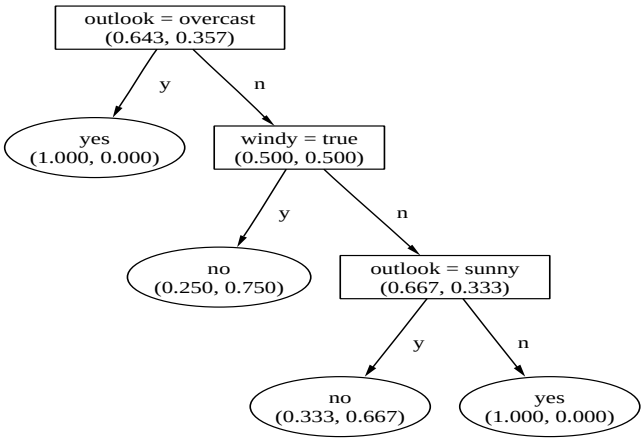

Figure 1: A sample decision tree for the weather dataset.

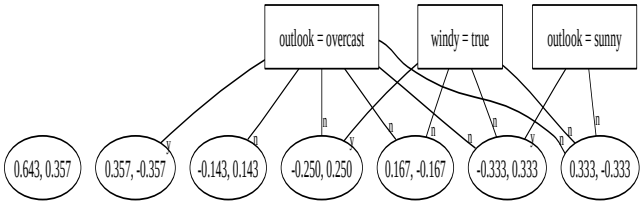

Figure 2: Transformed tree of Figure 1.

2.1 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.

A standard decision tree can be transformed into this representation. The following procedure is repeated for each node in the tree: Collect the conditions leading to the node—add any of these tests not already present to the top layer of the graph. To the bottom layer add the prediction value, which is the class distribution of the node minus the class distribution of its parent. Link the two layers, noting whether the condition must be satisfied as true or false for the values to apply.

Figure 2 shows the result of transforming the tree in Figure 1 to the flattened form.

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 a new prediction value has 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 key to merging the graphs efficiently. Nodes sharing identical preconditions will be merged independent of the order of these preconditions in the original trees.

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 tree making up the model.

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.

Trees can be merged together by making use of the flat representation. This is accomplished by using the procedure described in the last section repeatedly, adding multiple trees to the same model.

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 algorithm 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.

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 3 shows Figure 2 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.

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.

Start with (0.643, 0.357)
Add (0.357, -0.357) if $outlook = overcast$.
Add (-0.143, 0.143) if $outlook! = overcast$.
Add (-0.250, 0.250) if $outlook! = overcast$ and $windy = true$.
Add (0.167, -0.167) if $outlook! = overcast$ and $windy! = true$.
Add (-0.333, 0.333) if $outlook! = overcast$ and $windy! = true$ and $outlook = sunny$.
Add (0.333, -0.333) if $outlook! = overcast$ and $windy! = true$ and $outlook! = sunny$.
Sum is final voting distribution for classes (yes, no).

Figure 3: Voting rules derived from Figure 2. The voting weights correspond to the labels in Figure 2.

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.

2.4 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 model's 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, 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 4 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 1000 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 degradation of random removal will fall somewhere between the two extremes of largest-first and smallest-first, as is demonstrated.

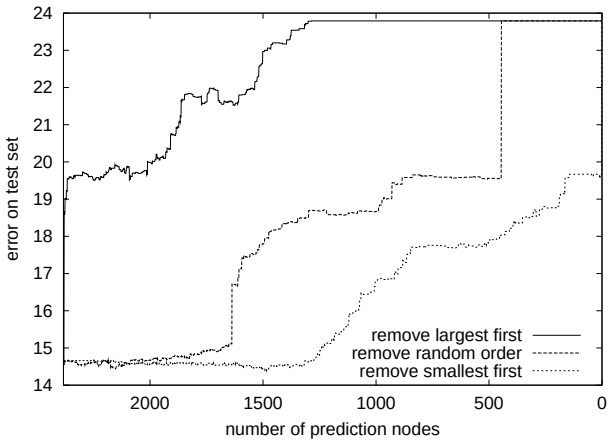

Figure 4: Effect removal order has on test set accuracy when pruning back a full model trained on the adult dataset.

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 5: Chunking algorithm.

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.

In the experiments reported below we will see that this straightforward pruning strategy can degenerate, especially for small global memory sizes. To counter this behaviour we introduce the notion of a "bouncer" which is a simple safeguard checking whether the merged and pruned model actually performs better on the current batch than the initial model. If not, the initial model is retained. Figures 13 to 15 show the positive effect of using such a safeguard.

2.5 Chunking

Figure 5 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 decision tree learning algorithm to induce a decision 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 two parameters that can be controlled—the chunk size and the maximum number of prediction values allowed in the model. The chunk size has an effect on the run-time complexity of the algorithm. Under the real-time online constraint, for a super-linear algorithm, one would set this parameter to ensure that learning time can keep pace with the data flow. The second parameter puts a cap on the memory usage of a model, and therefore also on the prediction time, at the potential expense of accuracy.

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

3.1 Datasets and Methodology

Our choice of datasets is rather limited for the following reasons: the pruning strategy described in the previous chapter is currently only applicable to two-class problems; we want to have large enough datasets to be able to study the effects of batch-incrementality; and we also want to compare to learning from the full dataset at once. Therefore datasets cannot be too large in this experimental evaluation. Datasets are either from UCI [3] or synthetically generated. To benchmark our method versus other methods, we use the WEKA [1] implementation of C4.5 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 6 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

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

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

Table 1: Datasets used for the experiments.

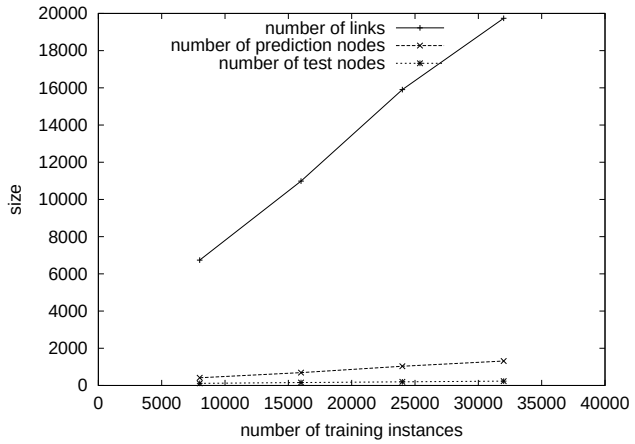

Figure 6: Model growth on the adult dataset (chunk size 8000).

of test nodes—this is where most replication occurs. So without pruning the global model would grow unboundedly for infinite data streams.

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 [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 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. 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 minimum—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 minimum for a given dataset is a topic for further research. One possibility is to race several candidates as in [7]. In this paper we only investigate the effect on accuracy that a small number of different fixed chunk sizes have.

Figures 7 through 9 show the effect on prediction accuracy when the chunk size is varied on three datasets.

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 severely 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 up the chunk size too high means the data is quickly exhausted without providing an appreciation of trends over time.

Fixing the chunk size to 8000, we investigate the effect of pruning on the accuracy of the model. In Figures 10 through 12 we restrict the number of prediction nodes allowed in the model, using the pruning technique outlined in Section 2.4, to study 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.

Additionally, after a certain number of batches have been merged, performance can deteriorate to a point worse than the performance of the very first batch. Currently we have no satisfactory explanation for this phenomenon, but we assume that is related to small disjuncts [10] whose respective rules might be pushed out of the global model by strong rules from later batches which are variants of strong rules already present. This observation has triggered the introduction of the pruning safeguard as described in the previous section. Figures 13 to 15 show the effect this safeguard has on pruning. Even though it does not always pinpoint the optimal spot, it seems very successful in keeping the accuracy of especially very small models close to the optimum. Still, it is only a first step, and there may be better ways of both pruning and safeguarding.

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

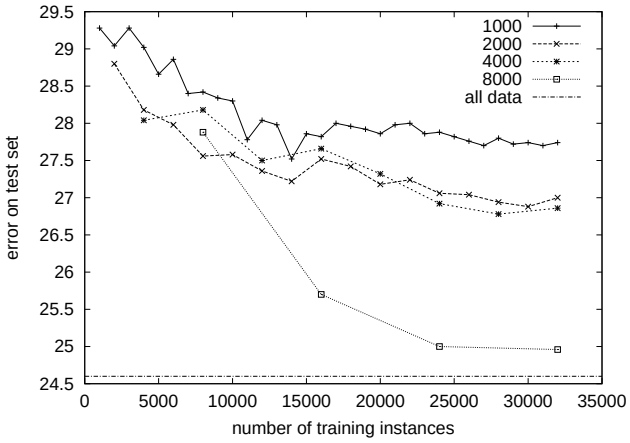

Figure 7: Varying chunk size on anonymous.

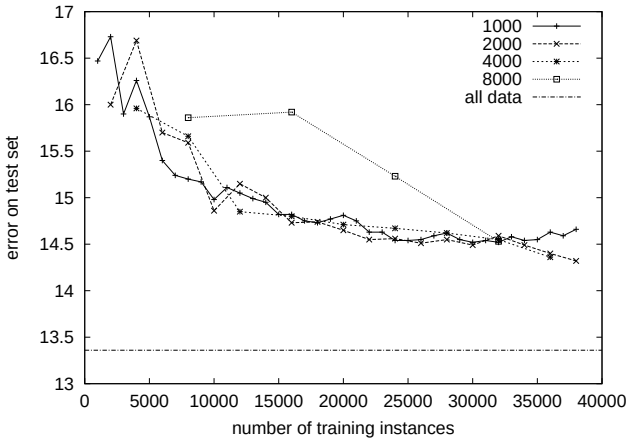

Figure 8: Varying chunk size on adult.

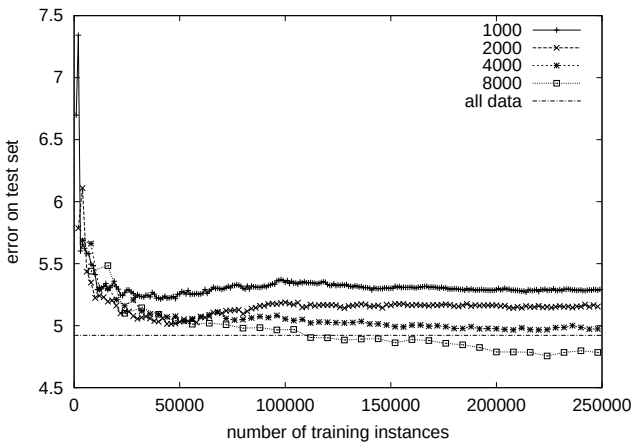

Figure 9: Varying chunk size on census-income.

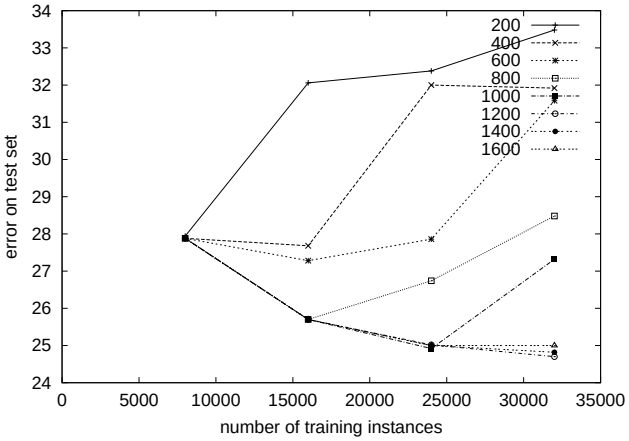

Figure 10: Varying maximum model size on anonymous.

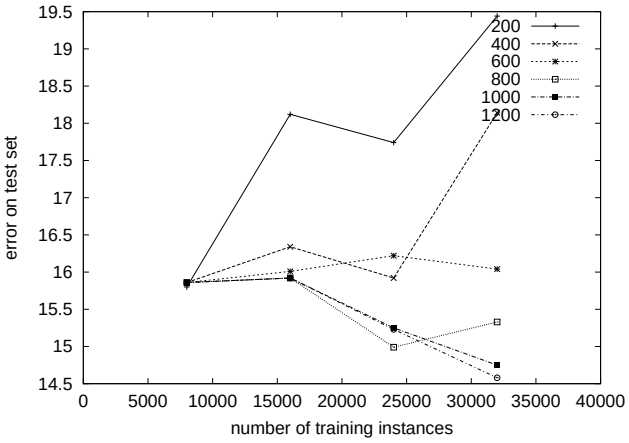

Figure 11: Varying maximum model size on adult.

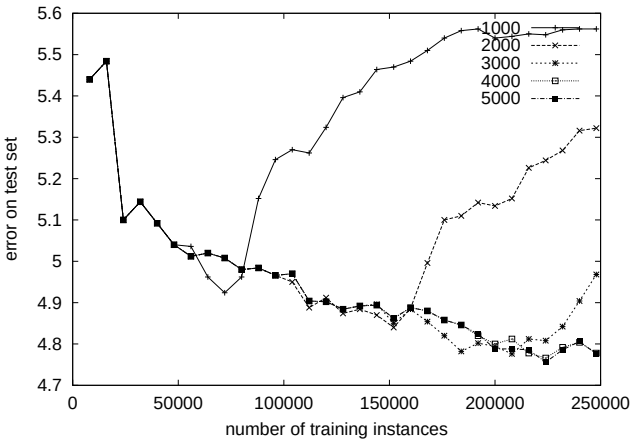

Figure 12: Varying maximum model size on census-income.

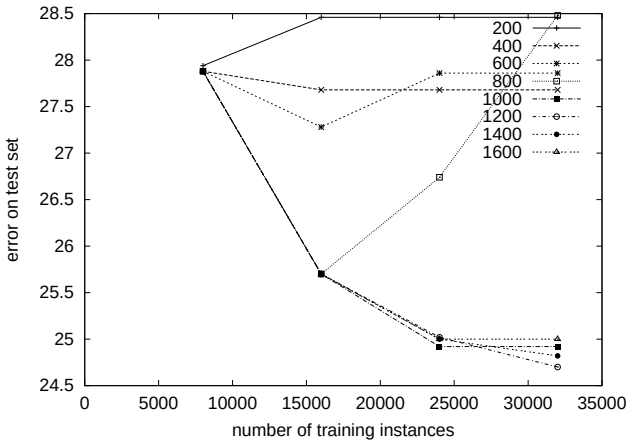

Figure 13: Varying maximum model size on anonymous plus safeguarding.

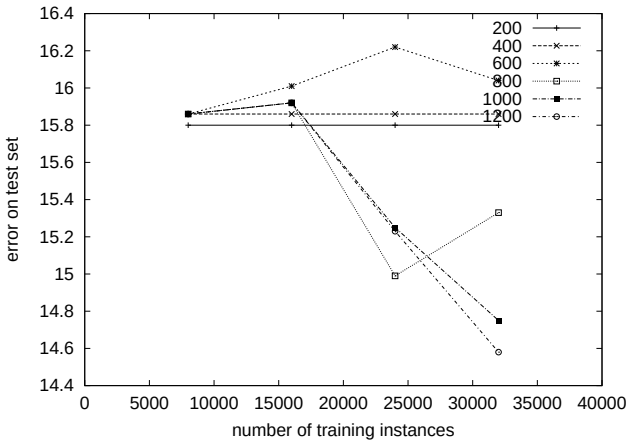

Figure 14: Varying maximum model size on adult plus safeguarding.

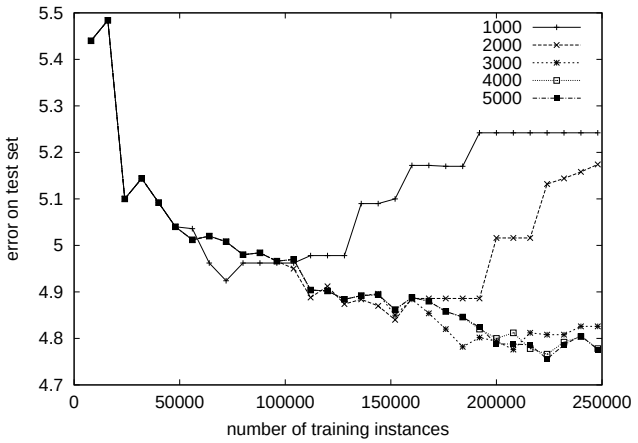

Figure 15: Varying maximum model size on census-income plus safeguarding.

Dataset	C4.5	BIC	BBIC
anonymous	24.60%	24.96%	24.92%
adult	13.36%	14.53%	15.33%
census-income	4.92%	4.78%	4.83%

Table 2: Error on test set.

Dataset	C4.5	BIC	BBIC
anonymous	611	1767	1000
adult	483	1315	800
census-income	2041	5405	3000

Table 3: Number of rules in model.

Table 2 compares the accuracy of our technique against learning from all the data at once. The first column shows the error obtained by training C4.5 on the entire training set. The second column shows the error for a batch-incremental learning (BIC) without any limit on the size of the final model. The BBIC (bounded batch-incremental classifier) column shows the error obtained using our chunking algorithm using a chunk size of 8000, and a limit on the global model size (as evident from Table 3), as well as the pruning safeguard.

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 for BIC and BBIC. The size of the BIC models is significantly larger than the size of the C4.5 induced decision trees.

Table 2 contains some interesting results. First, C4.5 is only significantly better on the adult dataset. Second, BBIC is only performing significantly worse than BIC on one dataset, which again is the adult dataset. In all other cases all algorithms perform similarly. Significance was determined using McNemar's test [5], which is a reasonable test given predefined large test sets. Summarizing we can claim that the BBIC approach seems to be able to achieve accuracies comparable to standard approaches which need to keep all of the training data in main memory.

To investigate the scalability of our algorithm we use synthetically generated data. We use a chunk size of 8000 and limit the size of the model to 5000 prediction values. Figures 16 and 17 show the training and testing times when scaling from 1 to 5 million training instances. The testing time is the time it took to classify a constant number of instances after training to that point.

The figures show that training time scales linearly with the size of the data, and that testing time is constant once the model has reached its maximum size. Figure 18 shows the sizes of the models by counting the number of links between test nodes and prediction nodes. As the number of prediction nodes is fixed to 5000 and as test nodes can be shared between multiple rules, the number of links is probably the best indicator for model size behaviour. Once again, as expected, this behaviour is more or less constant.

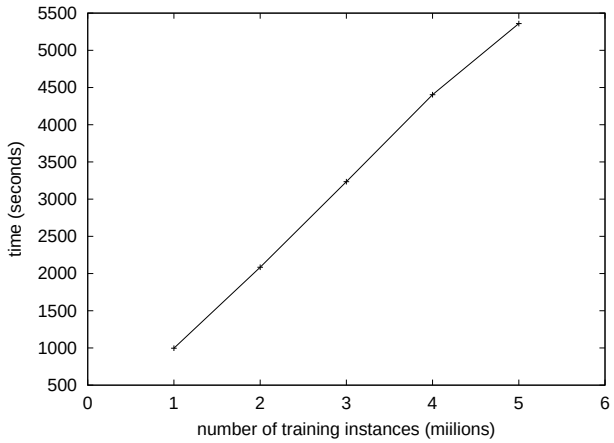

Figure 16: SynthF7 training times.

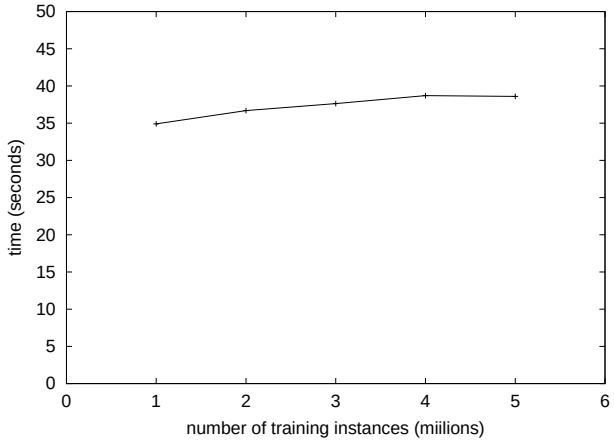

Figure 17: SynthF7 testing times.

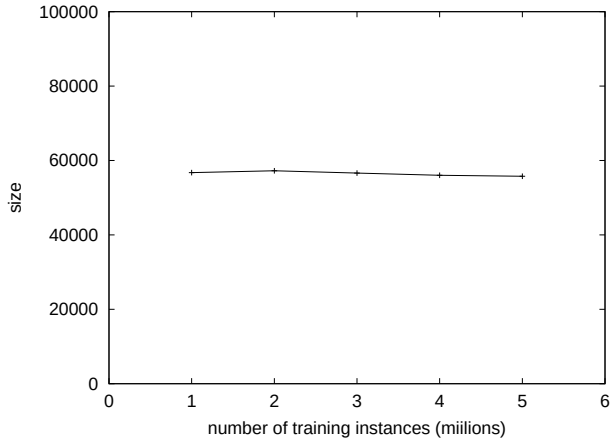

Figure 18: SynthF7 number of links.

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 [16]. CLOUDS [2] 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 [6], 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 [11].

The approaches most closely related to this paper use ensembles. Street and Kim propose SEA [17], in which models are bagged over chunks of a data stream and weighted. Frank et al. [7] boost subsequent models and prune those that fail to improve performance. Chunk sizes are raced to determine optimal size. In [20] 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. Additionally, none of these methods can guarantee that the generated model will stay within a pre-specified constant maximum size.

5. CONCLUSIONS

We have presented an algorithm for mining data streams using decision trees in a batch-incremental setting. The method has two correlated parameters, chunk size and model size. Experiments have been performed to observe the influence these parameters 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 larger chunk sizes will give rise to better classification performance, as will larger global model sizes. 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 time in the batch-incremental setup is linear in the number of instances fulfilling the scalability requirements of the data stream model. Merging the models of each batch into one global model and the subsequent pruning of this global model ensures constant memory requirements.

The presented algorithm therefore is a realistic solution to the problem of mining data streams. 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, developing more sophisticated pruning strategies, tracking concept drift, and dealing with multi-class problems.

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

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

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

[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] R.C. Holte, L.E. Acker, and B.W.W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[11] 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.

[12] Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In *National Conference on Artificial Intelligence*, pages 223–228, 1992.

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

[16] 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.

[17] 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.

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

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

[20] 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.