Stress-Testing Hoeffding Trees

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. Hoeffding trees are state-of-the-art in classification for data streams. They perform prediction by choosing the majority class at each leaf. Their predictive accuracy can be increased by adding Naive Bayes models at the leaves of the trees. By stress-testing these two prediction methods using noise and more complex concepts and an order of magnitude more instances than in previous studies, we discover situations where the Naive Bayes method outperforms the standard Hoeffding tree initially but is eventually overtaken. The reason for this crossover is determined and a hybrid adaptive method is proposed that generally outperforms the two original prediction methods for both simple and complex concepts as well as under noise.

1 Introduction

The Hoeffding tree induction algorithm [2] has proven to be one of the best methods for data stream classification. Standard Hoeffding trees use the majority class at each leaf for prediction. Previous work [3] has shown that adding so-called functional (or Naive Bayes) leaves to Hoeffding trees for both synthetically generated streams, and real datasets of sufficient size, as well as in the presence of noise outperforms standard Hoeffding trees.

In this paper we use an experimental evaluation methodology for data streams, where every single instance in the stream is used for both learning and testing. Using this methodology and an order of magnitude more data we discover situations were the standard Hoeffding tree unexpectedly outperforms its Naive Bayes counterpart. We investigate the cause and propose modifications to the original algorithm. An empirical investigation compares these modifications to both the standard Hoeffding tree and its Naive Bayes variant, and shows that one of the possible modifications is very robust across all combinations of concept complexity and noise. The modifications *only* concern the method of prediction. The standard Hoeffding tree learning algorithm is used in all cases.

The paper is arranged as follows. Section 2 contains an evaluation of Hoeffding trees with an unexpected result. Section 3 proposes several solutions to address the problem, and Section 4 evaluates and discusses them. Finally Section 5 concludes the paper.

A. Jorge et al. (Eds.): PKDD 2005, LNAI 3721, pp. 495–502, 2005.
© Springer-Verlag Berlin Heidelberg 2005

2 Examining Hoeffding Trees

Data streams present unique opportunities for evaluation, due to the volume of data available and the any-time property of the algorithms under examination. We consider a method of evaluation that exploits this property whilst maximizing use of the data. This is achieved by using every instance as a testing example on the current model before using it to train the model, incrementally updating statistics at each point.

The particular implementation of Hoeffding Tree induction discussed in this paper uses information gain as the split criterion, the original VFDT Hoeffding bound formulation [2] to determine when to split (using parameters $\delta = 10^{-6}$, $\tau = 5\%$, and $n_{min} = 300$), and handles numeric attributes by Gaussian approximation (throughout **ht** refers to this algorithm and **htnb** the same algorithm with Naive Bayes prediction at the leaves).

Our first analysis looks at the difference in accuracy between **ht** and **htnb**. We start with data generated by a randomly constructed decision tree consisting of 10 nominal attributes with 5 values each, 10 numeric attributes, 2 classes, a tree depth of 5, with leaves starting at level 3 and a 0.15 chance of leaves thereafter (the final tree had 741 nodes, 509 of which were leaves)—which we shall refer to as the simple random tree. Note that for all graphs in this paper we have averaged over 10 runs to eliminate order effects.

Figure 1 shows the result of evaluating over 10 million instances with no noise present. As in previous studies, it is clear that **htnb** gives an improvement in classification accuracy, with both variants performing well, reaching around 99% accuracy in the long run.

Figure 2 shows the impact that noise has. 10% noise was introduced to the data with uniform randomness. A different picture emerges—**htnb** looks better initially but somewhere before 2 million instances the graphs cross over and in the long run **htnb** is worse.

Next, the tree generator is adjusted to produce a complex random tree—50 nominal attributes with 5 values each, 50 numeric attributes, 2 classes, a tree

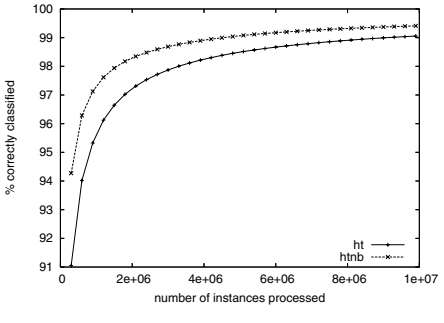

Fig. 1. Simple random tree generator with no noise

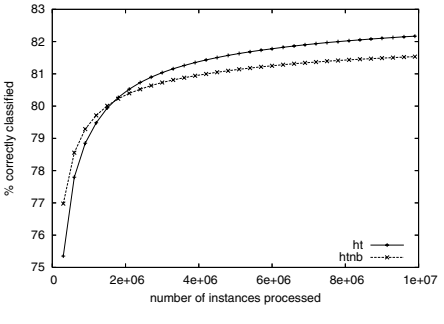

Fig. 2. Simple random tree generator with 10% noise

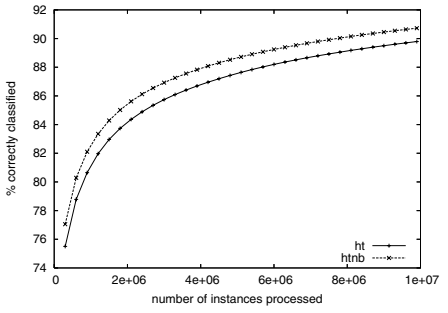

Fig. 3. Complex random tree generator with no noise

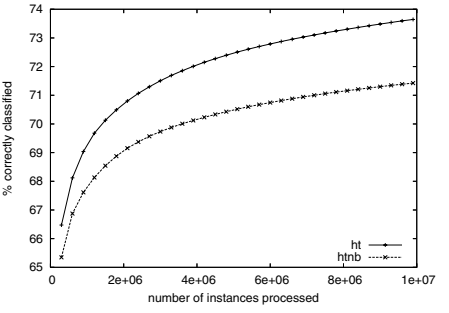

Fig. 4. Complex random tree generator with 10% noise

depth of 10, with leaves starting at level 5 and a 0.15 chance of leaves thereafter (the final tree had 127,837 nodes, 90,259 of which were leaves).

Figures 3 and 4 show the learning curves resulting from the complex random tree data, both clean and noisy. The same pattern is observed, only this time **htnb** is worse from the outset.

Testing the simple tree with other noise levels gives result similar to Figure 2, at 5% noise the crossover occurs later, and at 20% it occurs earlier than the 10% case. On the complex tree data the gap in Figure 4 widens as the noise increases.

3 Possible Solutions to the Problem

The evaluation uncovered cases where **htnb** is less accurate than **ht**. This behaviour appears when noise is introduced, and becomes more pronounced when the concept is more difficult to learn. The problem shows up on other datasets also, though sometimes it is less apparent due to the millions of instances that are needed before crossover occurs.

We speculate that the problem is due to small disjuncts in the tree in combination with noisy data. The leaves see only an insufficient number of examples, and the ones they do see are noisy, causing **htnb** to be less accurate than **ht**.

Experiments with severely limited maximum tree sizes supported this hypothesis: **htnb** eventually outperformed **ht** once tree growth was artificially stopped.

If the problem with **htnb** under noise is due to the models being unreliable in their early stages, then there are several ways the problem could be solved.

One solution has been proposed by Gama who in [3] suggests the use of a short term memory cache housing some of the recently seen instances. A problem with this solution is determining a sufficient size for the cache. As the tree grows in complexity, fewer of the instances in the cache will be applicable to the new leaves deep in the tree (we refer to this as **htnb-stm**x where x is the cache size).

Another idea is to inherit information from the parent once a split has been chosen. For an attribute split, it is possible to approximate the distribution of

values resulting from the split. For the other attributes, less information is known about the result of the new split, but we can assume that the distribution is the same as in the parent. This approximation may be grossly incorrect, but at least it gives the model a starting point rather than starting with no information (**htnbp**).

A potentially serious problem with this approach is that if the statistics used to make split choices are primed then the split decisions will be altered, having an impact on the tree structure. As will be shown in Section 4, this generally has a detrimental effect on accuracy.

A solution to this is to maintain a separate model per leaf that is used for prediction purposes only, and leave the split decision statistics untouched. This effectively doubles the storage requirement per leaf (**htnbps**).

An adaptive solution is to see how often the Naive Bayes leaves make classification errors compared with choosing the majority class, using Naive Bayes leaves only when their measured accuracy is higher. The data stream setting affords us the ability to do this as we can monitor performance on unseen instances in the same way that the overall evaluation is performed (**htnba**).

The method works by performing a Naive Bayes prediction per training instance, comparing its prediction with the majority class. Counts are stored to measure how many times the Naive Bayes prediction gets the true class correct as compared to the majority class. When performing a prediction on a test instance, the leaf will only return a Naive Bayes prediction if it has been more accurate overall than the majority class, otherwise it resorts to a majority class prediction.

To complete the experimentation, we added priming and model separation to **htnba**, these are referred to as **htnbap** and **htnbaps** respectively.

Table 1 summarizes the costs associated with the candidates, beyond that needed for **htnb**. The costs associated with the adaptive choice are minor—a few extra counts and a single comparison per prediction. The Naive Bayes prediction per training instance is a cost that can be shared with the evaluation mechanism. The costs associated with maintaining a separate prediction model are the greatest—effectively doubling the storage and update time per leaf. As splitting is a much less frequent operation than anything else, higher splitting costs usually do not have much impact on the overall total cost.

4 Results and Discussion

Figures 5 to 8 show the result of the various prediction strategies on the simple and complex tree data, with and without noise.

Figure 5 shows that the two methods of priming without separate models (**htnbp** and **htnbap**) do worse than **ht**. All other Naive Bayes methods outperform **ht** by roughly equal amounts. Introducing noise in Figure 6 sees **htnb** doing worst overall, even worse than **htnbp** and **htnbap**. The other methods (besides those using short term memory) successfully overcome the problem with little between them.

Table 1. Additional space/time costs beyond **htnb** requirements

	space per tree	space per leaf	time per training instance	time per test instance	time per split
htnb-stm$x$	cache of x instances		cache update		pass instances to leaves + NB updates
htnbp					distribution estimation
htnbps		NB model	NB update		distribution estimation
htnba		error count	NB prediction count update	decide MC or NB	
htnbap		error count	NB prediction count update	decide MC or NB	distribution estimation
htnbaps		error count NB model	NB prediction count update NB update	decide MC or NB	distribution estimation

Figure 7 explores the case of a more complex but still noise-free concept. Results are similar to the simple tree case (Figure 5). There is not much separation within the group of methods that outperform **ht**. Once again both **htnbp** and **htnbap** perform worse than **ht**.

Adding noise to more complex concepts results in the greatest separation between the techniques. Figure 8 shows the short term memory solution to be unsatisfactory. A short term memory of 1000 instances hardly does better than **htnb**, which is the worst performer. Increasing the cache size to 10000 instances does little to improve the situation.

Figure 8 demonstrates the superiority of the adaptive method. **htnbp** and **htnbps** fall short of **ht**, while all of the adaptive methods do better. The best performing method is also the most costly one (**htnbaps**), with the less expensive **htnba** not far behind.

To investigate whether these findings hold more generally, experiments on two additional UCI datasets [1] were conducted. The LED dataset is a synthetic generator allowing us to generate the desired 10 million instances. The particular configuration used of the LED generator produced 24 binary attributes, 10 classes, and 10% noise.

The results in Figure 9 exhibit slightly different looking curves. The majority of the methods hover around 26% error which is known to be the optimal Bayes error for this problem. The exceptions are **ht** (which has a much slower learning curve without the aid of Naive Bayes leaves), and **htnb**. The failure of **htnb** shows that the problem extends beyond tree generated data. The success of the others show that the problem can be alleviated.

These results contradict those reported by Gama et al. [3,4], whose conclusion was that Naive Bayes leaves are always better on the LED data. They used a

Table 2. Final accuracies achieved on tree generators

	simple tree no noise	simple tree 10% noise	complex tree no noise	complex tree 10% noise
ht	99.056 ± 0.033	82.167 ± 0.031	89.793 ± 0.168	73.644 ± 0.151
htnb	99.411 ± 0.026	81.533 ± 0.021	90.723 ± 0.153	71.425 ± 0.118
htnb-stm1k	99.407 ± 0.027	81.544 ± 0.019	90.768 ± 0.150	71.527 ± 0.108
htnb-stm10k	99.409 ± 0.025	81.593 ± 0.018	91.008 ± 0.153	71.658 ± 0.085
htnbp	97.989 ± 0.058	81.853 ± 0.042	88.326 ± 0.209	73.029 ± 0.121
htnbps	99.376 ± 0.028	82.456 ± 0.023	90.598 ± 0.153	73.063 ± 0.124
htnba	99.408 ± 0.027	82.510 ± 0.024	90.874 ± 0.153	74.089 ± 0.141
htnbap	98.033 ± 0.057	81.938 ± 0.040	88.609 ± 0.211	73.675 ± 0.127
htnbaps	99.375 ± 0.028	82.545 ± 0.024	90.935 ± 0.148	74.249 ± 0.134

Table 3. Final accuracies achieved on other datasets

	LED	Covertype
ht	72.851 ± 0.031	66.832 ± 0.163
htnb	71.645 ± 0.013	69.064 ± 0.135
htnbp	73.928 ± 0.005	68.476 ± 0.040
htnbps	73.799 ± 0.041	69.049 ± 0.145
htnba	73.935 ± 0.005	70.998 ± 0.087
htnbap	73.961 ± 0.004	71.388 ± 0.037
htnbaps	73.996 ± 0.005	71.054 ± 0.095

hold out test set and quoted the final accuracy attained. The largest training set used in their work was 1.5 million instances. In our experiment the problem does not occur until about 4 million instances.

Finally, the algorithms were tested on real data using the Forest Covertype dataset. This consists of 581,012 instances, 10 numeric attributes, 44 binary attributes and 7 classes. To do 10 runs over this data the instances were randomly permuted 10 different ways. In Figure 10 we see three distinct groups. The worst performer is **ht**. The next group consists of the non-adaptive methods **htnb**, **htnbp** and **htnbps**. The group of best performers are the adaptive ones. This result demonstrates that even in cases where **htnb** is not obviously underperforming, adding the adaptive modification can enhance performance.

Tables 2 and 3 show the final accuracies achieved along with the standard error for all of the graphs displayed in Figures 5 through 10.

Overall these results support the conclusion that priming the leaf models without using a separate model per leaf results in poor performance. Without the separate model, the split decisions are altered in such a way that the tree is less accurate. Inclusion of the separate model improves the situation (at a cost), but it appears not as helpful as using the adaptive method.

Our experiments demonstrate that **htnba** provides a good compromise between accuracy and cost. In some cases it did slightly worse than **htnbaps**, but

the difference does not justify the extra cost. The adaptive approach of **htnba** has a relatively low overhead, meaning it can be justified over **ht**, and especially over **htnb**, in all but the most extreme resource-bounded situations.

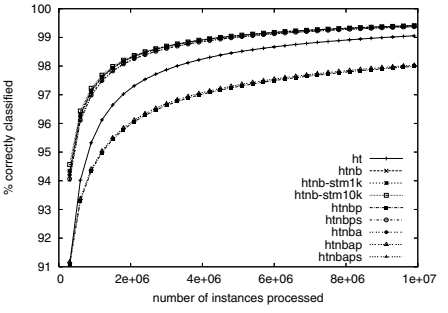

Fig. 5. Simple random tree generator with no noise

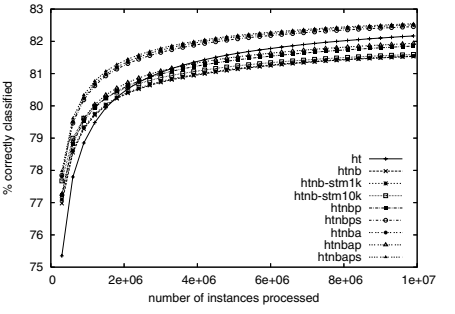

Fig. 6. Simple random tree generator with 10% noise

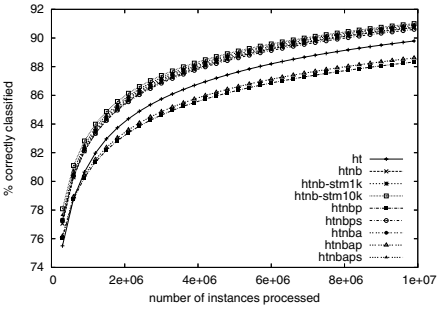

Fig. 7. Complex random tree generator with no noise

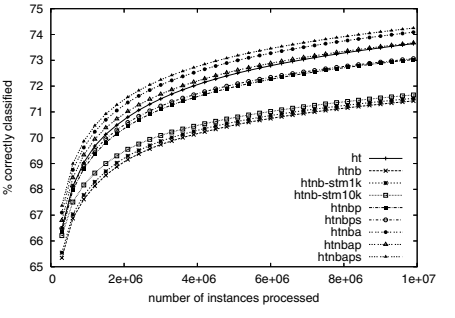

Fig. 8. Complex random tree generator with 10% noise

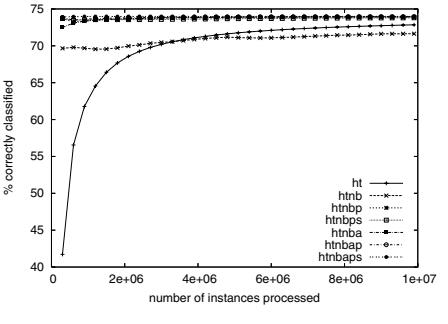

Fig. 9. LED generator

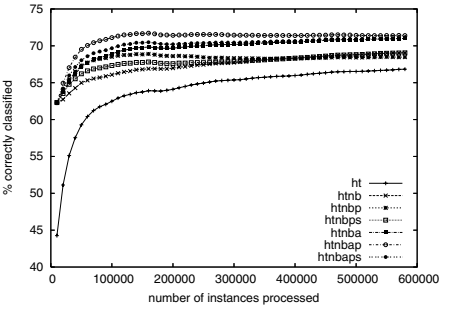

Fig. 10. Covertype

5 Conclusions

By using noise and synthetically generating large and complex concepts we have performed experiments that stress-test Hoeffding trees. Our focus has been on the two common prediction methods used in association with these trees: majority class prediction and Naive Bayes leaf prediction. In this experimental environment we uncovered an unexpected problem with Naive Bayes leaf prediction. Multiple improvements to the shortcomings of this method were invented and empirically evaluated. The best solution adaptively decides when the Naive Bayes leaves are accurate enough to be trusted. This adaptive method only imposes a minor additional cost on the algorithm, yet seems to almost guarantee equal or better accuracy than a simple majority class prediction.

References

1. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
2. Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
3. Joao Gama, Pedro Medas, and Ricardo Rocha. Forest trees for on-line data. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 632–636, New York, NY, USA, 2004. ACM Press.
4. Joao Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528, New York, NY, USA, 2003. ACM Press.