New Options for Hoeffding Trees

Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby

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

Abstract. Hoeffding trees are state-of-the-art for processing high-speed data streams. Their ingenuity stems from updating sufficient statistics, only addressing growth when decisions can be made that are guaranteed to be almost identical to those that would be made by conventional batch learning methods. Despite this guarantee, decisions are still subject to limited lookahead and stability issues. In this paper we explore Hoeffding Option Trees, a regular Hoeffding tree containing additional *option* nodes that allow several tests to be applied, leading to multiple Hoeffding trees as separate paths. We show how to control tree growth in order to generate a mixture of paths, and empirically determine a reasonable number of paths. We then empirically evaluate a spectrum of Hoeffding tree variations: single trees, option trees and bagged trees. Finally, we investigate pruning. We show that on some datasets a pruned option tree can be smaller and more accurate than a single tree.

1 Introduction

When training a model on a data stream it is important to be conservative with memory and to make a single scan of the data as quickly as possible. Hoeffding trees [6] achieve this by accumulating sufficient statistics from examples in a node to the point where they can be used to make a sensible split decision. A decision, in fact, that is guaranteed to be almost as reliable as the one that would be made by an algorithm that could see all of the data.

The sufficient statistics turn out to be beneficial for both tree growth and prediction as they can be used to build Naive Bayes models at the leaves of the tree that are more accurate than majority class estimates [7]. There remain situations where Naive Bayes leaves are not always superior to majority class, and [8] shows that adaptively deciding when their use is beneficial per leaf of the tree can further enhance classification performance.

Given this sequence of improvements to the basic algorithm, is it possible to improve predictive performance further? Contemporary machine learning would suggest that the next logical step is to employ ensembles of Hoeffding trees using a framework such as bagging or boosting.

In part, ensemble methods seek to overcome problems inherent in all greedy tree learners. Tree learners have limited lookahead ability (the best scoring single

M.A. Orgun and J. Thornton (Eds.): AI 2007, LNAI 4830, pp. 90–99, 2007.

test based on the children that test would generate) and they are not stable. Stability [2] relates to the selection of a particular attribute split when the scores of other attribute splits are close. The examples processed just prior to the split decision could have an undue influence over this decision and it may not be the best decision in the longer term. Breiman [4] demonstrates just how unstable this influence can be, especially when marginal decisions are taken near the root of the tree. Even though decisions are made with more data in a Hoeffding tree, they are still prone to these two issues.

The use of classifier ensembles for data streams has been explored before, most notably in the work on online bagging and boosting by Oza and Russell [10]. Of these though, only bagging appears to work successfully in practice. In addition to the important issue of ensemble model size, ensemble methods produce models that are not interpretable.

In the context of batch learning, these issues led several authors to develop option trees [5,9]. In a sense, option trees represent a middle ground between single trees and ensembles. They are capable of producing useful, and interpretable, additional model structure without consuming too many resources.

Option trees consist of a single structure that efficiently represents multiple trees. A particular example can travel down multiple paths of the tree, contributing, in different ways, to different options. The class of a test example is determined by a committee made up of the predictions of all leaf nodes reached. In the context of a data stream, the idea is to grow multiple options in the same manner as a single Hoeffding tree. By doing so the effects of limited lookahead and instability are mitigated, leading to a more accurate model.

2 Option Trees

Figure 1 is an example of what the top few levels of an option tree can look like. The tree is a regular decision tree in form except for the presence of option nodes, depicted in the figure as rectangles. At these nodes multiple tests will be applied, implying that an example can travel down multiple paths of the decision tree, and arrive at multiple leaves.

Algorithm 1 describes the Hoeffding option tree induction process. This process is a modification of the most recent and more robust version of the original

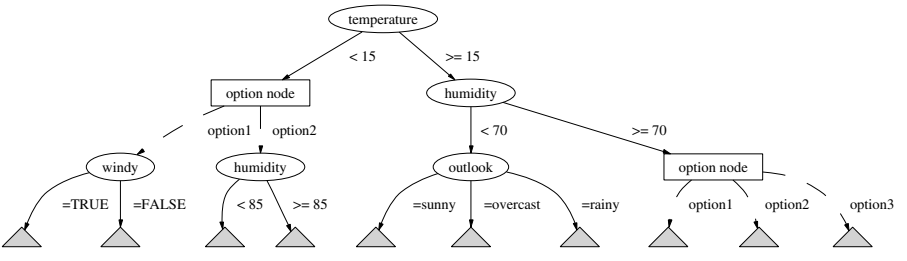

Fig. 1. An option tree

Hoeffding tree induction algorithm [8] to allow exploration of multiple options. Lines 19-34 enable introduction of options into the tree, removing these lines results in the standard Hoeffding tree algorithm, as will setting *maxOptions* to 1. The other differences between this algorithm and the standard one occur in lines 3 and 4, where an example updates multiple option nodes down multiple paths—in the original algorithm only a single leaf will be influenced by each particular example. As is common with standard Hoeffding trees, the evaluation criterion, $\overline{G}$, for all test nodes is information gain. The implementation of this algorithm is referred to as HOT in subsequent discussions (see Section 4).

Predictions by the Hoeffding option tree are made by combining the predictions of all leaves applying to an example. Kohavi and Kunz [9] used majority voting, but like [5] we used weighted voting—experiments suggested that a weighted vote, where the individual probability predictions of each class are summed, performs better than unweighted voting.

2.1 Managing Tree Growth

Option trees have a tendency to grow very rapidly if not controlled. In [9] the authors employ several strategies to alleviate this problem, including limiting the number of options allowed locally per node. Our primary strategy for controlling growth limits the number of options allowed *per example*, placing a global limit that is less trivial to enforce but more effective at regulating growth. Every option node in the tree has a counter (*optionCount*) that is initialized to 1. As the tree grows the counters are updated to reflect the number of leaves reachable by an example at each node. This is not trivial—the number of nodes reachable at a particular node is determined by options found in both its ancestors and descendants. Tree growth is controlled by only allowing an option to be introduced where the counter has not reached the maximum allowable value, which guarantees a maximum number of paths that an example can follow down the tree. Equally this limits the number of leaves that will be involved in an update or prediction.

A range of option limits were tested and averaged across all datasets to determine the optimal number of paths. Figure 2 shows the result. As can be seen, prior to five paths, significant gains in accuracy are apparent, but after that point the accuracy gains diminish. Interestingly, this is exactly the same number of options used by [9] although they arrived at this figure by chance.

2.2 Restricting Additional Splits

In order to generate new structure at an option node we only consider options for attributes that have not already been split at that node. This further cuts down the eagerness of the tree to grow, and overcomes the problem of the same split being repeatedly chosen, resulting in multiple redundant paths.

The final element of tree growth management involves deciding when additional options look sufficiently competitive with existing splits in the tree to

Algorithm 1 Hoeffding option tree induction algorithm, where n_{min} is the grace period, $\overline{G}$ is the split criterion function, R is the range of $\overline{G}$, τ is the tie-breaking threshold, δ is the confidence for the initial splits, δ' is the confidence for additional splits and $maxOptions$ is the maximum number of options reachable by a single example

Let HOT be an option tree with a single leaf (the root)
for all training examples **do**
  Sort example into option nodes L using HOT
  for all option nodes l of the set L **do**
    Update sufficient statistics in l
    Increment n_l, the number of examples seen at l
    if $n_l \; mod \; n_{min} = 0$ and examples seen at l not all of same class **then**
      if l has no children **then**
        Compute $\overline{G}_l(X_i)$ for each attribute
        Let X_a be attribute with highest $\overline{G}_l$
        Let X_b be attribute with second-highest $\overline{G}_l$
        Compute Hoeffding bound $\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n_l}}$
        if $\overline{G}_l(X_a) - \overline{G}_l(X_b) > \epsilon$ or $\epsilon < \tau$ **then**
          Add a node below l that splits on X_a
          for all branches of the split **do**
            Add a new option leaf with initialized sufficient statistics
          end for
        end if
      else
        if $l.optionCount < maxOptions$ **then**
          Compute $\overline{G}_l(X_i)$ for existing splits and (non-used) attributes
          Let S be existing child split with highest $\overline{G}_l$
          Let X be (non-used) attribute with highest $\overline{G}_l$
          Compute Hoeffding bound $\epsilon = \sqrt{\frac{R^2 ln(1/\delta')}{2n_l}}$
          if $\overline{G}_l(X) - \overline{G}_l(S) > \epsilon$ **then**
            Add an additional child option to l that splits on X
            for all branches of the split **do**
              Add a new option leaf with initialized sufficient statistics
            end for
          end if
        else
          Remove attribute statistics stored at l
        end if
      end if
    end if
  end for
end for

warrant their introduction. The δ parameter controls the allowable error in decisions that occur during Hoeffding tree induction. The smaller this value, the longer the tree will wait before committing to a decision to split.

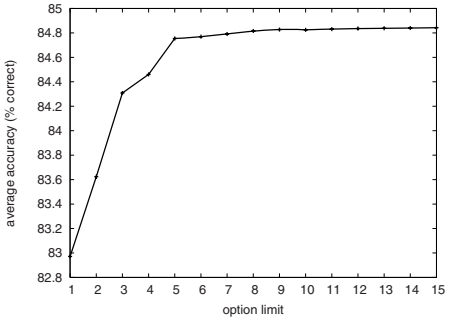

Fig. 2. The average accuracy achieved over all datasets by HOT varying the per-example option limit. Accuracies were estimated using an interleaved test-then-train evaluation over one million examples.

We introduce a second parameter, δ', for deciding when to add another split option beneath a node that already has been split. A new option can be introduced if the best unused attribute looks better than the current best existing option according to the $\overline{G}$ criterion and a Hoeffding bound with confidence δ'. Tie-breaking is not employed for secondary split decisions, so new options will only be possible when a positive difference is detected. δ' can be expressed in terms of a multiplication factor α, specifying a fraction of the original Hoeffding bound:

$$\delta' = e^{\alpha^2 ln(\delta)} \tag{1}$$

For example, with our default parameter settings of $\delta = 1E^{-8}$ and $\alpha = 0.05$ (that is, decisions are 20 times more eager), then from (1) $\delta' \approx 0.955$.

3 Pruning

Option trees improve accuracy at a cost: they consume more memory, as they can grow more rapidly than a standard Hoeffding tree. Not all that additional growth is necessarily beneficial, some subparts of the tree may be redundant, others can negatively impact accuracies. Pruning needs mechanisms for estimating the merit of subtrees and ways of selecting candidates for pruning. In a stream setting an additional issue is choosing an appropriate frequency for pruning as making decisions after every single example might be too time-consuming.

We implemented a pruning method modeled after reduced error pruning, where accuracies of subtrees were estimated by recording accuracy for every example seen at each node, and then periodically subtrees would be removed when their estimated accuracy appeared worse than their parents' accuracy. This method did not distinguish between standard splits and option nodes in the tree. It did not fare well, which we conjecture is due to its local nature, which cannot take into account the global effect averaging multiple options has:

on average a node may look worse than its parent, but it might be less correlated with other options and therefore help achieve the correct global prediction more often than the parent node.

Another approach tried to estimate the global contribution of each option. Every node in the tree with multiple options records accuracy estimates for each option for two settings: accuracy when including this option in global voting, and accuracy when excluding this option from global voting. Once estimated "exclusion" accuracy of an option exceeds the "inclusion" estimate, it (as well as all the substructure rooted at that option) becomes a candidate for pruning. The HOTP algorithm evaluated later in this paper implements this strategy. Every training example is used to update these accuracy estimates, and periodically (after every 100,000 examples) any option that appears to reduce accuracy is removed. Note that this pruning method is specialized and limited to options.

Unfortunately the results in Section 4 highlight that this way of pruning is not often very effective, either. Even though it is often able to reduce the size of the Hoeffding option trees, sometimes back to the size of the baseline Hoeffding tree, this reduction in size at times also causes a loss in accuracy. We have also, unsuccessfully, tried to base the pruning decision on a Hoeffding bound as well to make pruning more reliable.

We conclude that automatic pruning of Hoeffding option trees is very unlikely to produce satisfactory results, but also that pruning can improve performance on selected problems, given careful selection of the right method and parameter settings.

4 Empirical Evaluation

We use a variety of datasets for evaluation, most of which are artificially generated to make up for the lack of large publicly available real-world datasets. Table 1 lists the properties of the datasets.

Table 1. Dataset characteristics

name	nominal	numeric	classes	name	nominal	numeric	classes
GENF1-F10	6	3	2	WAVE21	0	21	3
RTS	10	10	2	WAVE40	0	40	3
RTC	50	50	2	RRBFS	0	10	2
LED	24	0	10	RRBFC	0	50	2
COVERTYPE	44	10	7				

The class functions and the generation processes for GENF1 through to GENF10 are described in [1]. RTS and RTC are simple and complex random tree concepts, based on the generation method described in [6], with 10% added noise. RTS has a maximum 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. RTC has a depth of 10 with leaves starting a level 5; the final tree had 127,837 nodes,

90,259 of which were leaves. Each nominal attribute has 5 possible values. LED, WAVE21 and WAVE40 (waveform without/with noise), and COVERTYPE(CTYPE) are from the UCI [3] collection of datasets. RRBFS refers to a simple random RBF (Radial Basis Function) dataset comprising 100 centers, 10 attributes and 2 classes. RRBFC is more a complex version comprising 1000 centers, 50 attributes and 2 classes.

Table 2. Comparison of algorithms. Accuracy is measured as the final percentage of examples correctly classified over the 10 million test/train interleaved evaluation. Size for HT is measured in terms of the number of tracking nodes, and sizes for HOT, HOTP and BAG10 are measured relative to this. Time is measured as the time per prediction of the final tree relative to HT. The best individual accuracies are indicated in boldface.

	HT		HOT			HOTS		HOTP			BAG10		
dataset	acc	size	acc	size	time	acc	time	acc	size	time	acc	size	time
GENF1	95.03	2031	95.03	1.73	1.10	**95.04**	1.06	95.03	1.09	0.99	95.03	9.57	5.76
GENF2	83.53	3205	93.06	2.73	1.84	93.07	1.61	93.18	0.91	0.91	**94.02**	8.0	6.36
GENF3	97.51	604	97.51	1.19	1.01	97.51	1.00	97.51	1.12	1.01	97.51	10.08	6.5
GENF4	94.37	2823	94.37	2.56	1.42	94.39	1.28	94.27	1.18	1.04	**94.49**	10.32	8.29
GENF5	89.73	3131	92.15	4.80	2.40	92.06	2.10	92.01	1.26	1.04	**92.56**	12.07	8.15
GENF6	92.31	3390	92.85	3.06	1.68	92.87	1.36	92.26	0.94	1.01	**93.13**	9.37	7.84
GENF7	96.62	2417	96.65	4.05	2.39	96.71	1.88	96.15	0.40	1.50	**96.77**	8.12	8.56
GENF8	99.39	563	99.39	2.90	1.61	99.40	1.25	99.38	1.61	1.06	99.40	5.41	6.15
GENF9	96.41	3270	96.46	3.81	2.35	96.50	1.79	95.47	0.55	0.92	**96.67**	6.18	8.46
GENF10	99.87	242	99.87	1.95	1.09	99.87	1.10	99.87	0.98	1.00	99.87	4.14	6.91
RTS	76.81	5013	77.09	2.05	1.14	77.09	1.14	76.91	1.16	1.03	**77.32**	10.31	3.47
RTC	58.84	3881	**61.51**	4.44	1.25	61.50	1.46	57.20	0.07	0.97	59.05	10.44	2.08
LED	73.99	302	73.99	4.71	2.97	73.98	1.75	73.99	0.38	2.04	**74.01**	10.03	6.16
WAVE21	84.90	1451	85.79	4.90	3.00	85.76	2.91	85.56	0.68	2.48	**85.86**	10.13	6.49
WAVE40	84.74	1463	85.68	4.88	2.55	85.69	2.73	85.44	0.53	1.98	**85.85**	9.98	4.95
RRBFS	90.50	2596	91.64	4.78	3.67	90.73	2.31	88.05	0.19	1.34	**92.89**	10.05	9.56
RRBFC	95.73	3101	97.84	4.86	3.21	97.13	2.37	93.64	0.08	3.11	**98.98**	9.68	7.0
CTYPE	70.92	41	71.53	3.39	1.83	71.11	1.92	71.50	3.24	1.83	**72.70**	9.63	4.69
average	87.84	2196	89.02	3.49	2.03	88.91	1.72	88.19	0.91	1.40	89.23	9.48	6.52

The evaluation methodology used every example for testing the model before using it to train. This interleaved test followed by train procedure was carried out on 10 million examples[1]. Table 2 reports the final accuracy, size and speed of the classification models induced. Additionally, the learning curves and model growth curves for selected datasets are plotted (Figure 3). For some datasets the differences in accuracy, as seen in Table 2, are marginal.

Sizes of the tree models are measured by counting the number of *tracker* nodes, i.e. nodes which store sufficient statistics. This measure is proportional to the actual size of the tree in memory, as the sizes of other data structures are insignificant compared to these nodes. In Table 2 the sizes of HOTS (see below)

[1] For COVERTYPE, evaluation was limited to the 581,012 examples available.

are omitted as they are equivalent, by definition, to the size of the corresponding single Hoeffding tree. Prediction speeds of the final models were measured by timing how long it took the model to classify an additional 1 million examples after training was complete. The prediction time figures in Table 2 are relative to the speed of HT[2].

The first, and baseline, algorithm (HT) is a single Hoeffding tree, enhanced with adaptive Naive Bayes leaf predictions. Parameter settings are $n_{min} = 1000$, $\delta = 1E^{-8}$ and $\tau = 0.05$. The second algorithm, HOT, is the Hoeffding option tree algorithm, restricted to a maximum of five option paths. Decisions to add more options to the tree were twenty times more eager than the initial decisions, so, as explained in Section 2.2, the parameter δ' was set to 0.955. HOTS is a modified version of HOT that halts growth once it reaches the same size as the final HT model on the same problem. The statistics in the leaves of the tree continue to be updated, but no additional nodes are added to the tree. This tests whether the accuracy advantages of option trees are solely due to higher memory consumption. HOTP tests one of the more successful attempts at pruning HOT. Every 100,000 examples, option paths are removed if it is estimated that they reduce the overall accuracy of the tree. BAG10 is online bagging [10] using ten Hoeffding trees.

Bagging is clearly the best method in terms of accuracy. This superior position is, however, achieved at high cost in terms of memory (almost ten times more memory is used on average) and prediction speed (six times slower on average). All option tree variants give superior accuracy performance over HT on average, with the options slowing down the prediction speed by at most a factor of two. HOT is more accurate than HT in all cases except LED[3], and sometimes the differences are substantial. This shows the potential of options to overcome issues seen in single trees due to instability and limited lookahead.

Options do not need additional memory to be useful: both HOTS and HOTP use essentially as much memory as a single tree, but still outperform it on average. Of the two, HOTS is the more impressive, performing at the level of the full option tree. It's prediction times are often close to the single tree and at most three times worse. Although HOTP returns good average performance, there are occasions where it does poorly. Its prediction times can be up to three times as high as HT, even though tree sizes are much smaller. This is caused by the presence of multiple options very close to the root of the HOTP tree.

Figure 3 shows the graphs where the option methods gain substantially over the single tree, except for HOTP on RTC, WAVE40, and RRBFS where vigorous pruning prevents the tree from reaching a reasonable size. On the RTC dataset both HOT and HOTS are superior to BAG10. HOTS uses only a little more memory than the single tree and approximately 10% of the ensemble memory.

[2] The actual speed for HT varied from around 4,000 predictions/second (for RTC) to 129,000 predictions/second (for GENF3).

[3] On LED, due to rounding this difference is not apparent in Table 2. All five algorithms are very close to the optimal Bayes error of 26%.

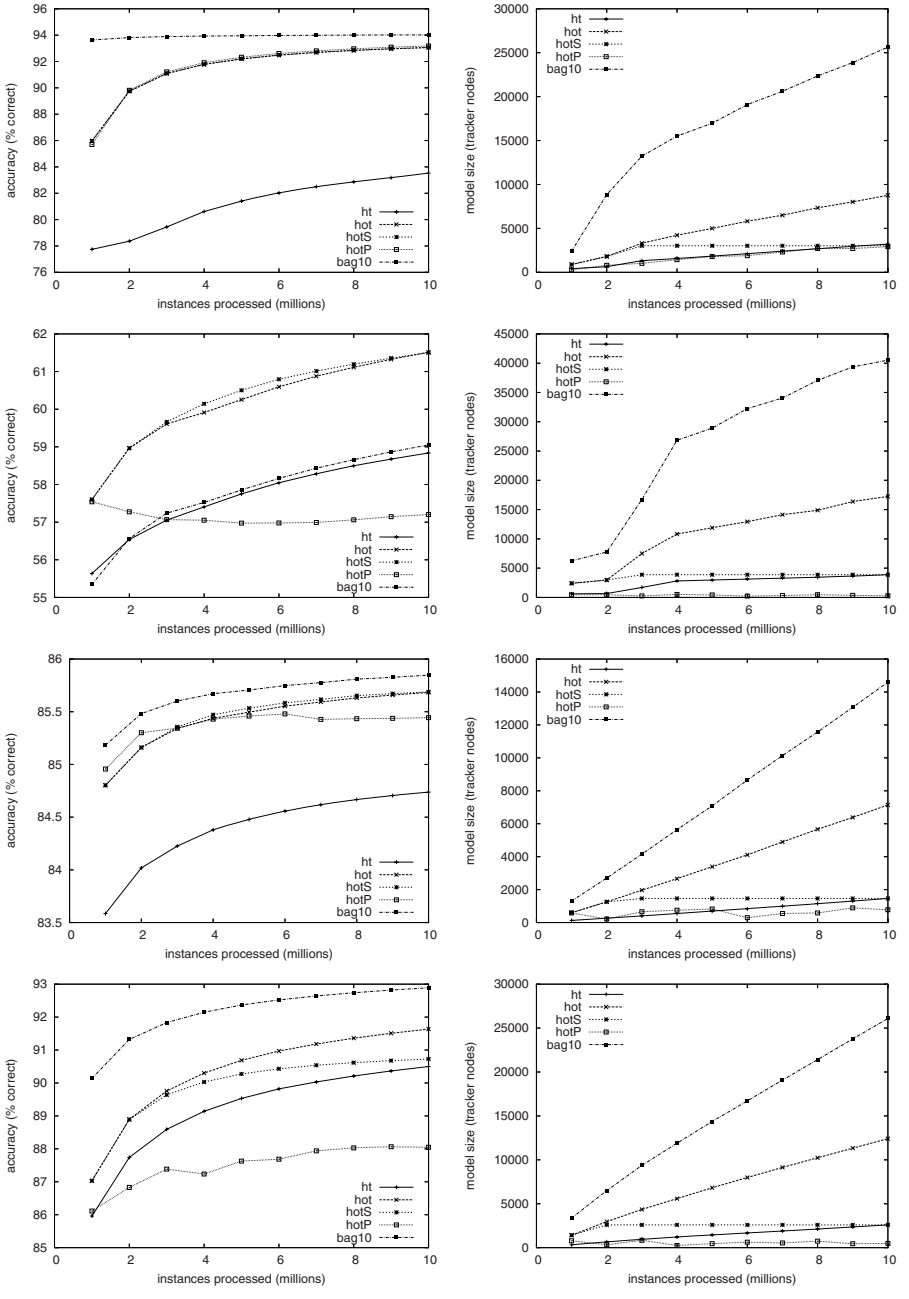

Fig. 3. Accuracy and tree size on (from top to bottom) GENF2, RTC, WAVE40 and RRBFS

5 Conclusions

We have demonstrated the efficacy of incorporating multiple paths via option nodes in Hoeffding trees. We described a method for controlling tree growth, and determined a reasonable number of options to explore. In all but one of our datasets the additional structure improved the performance of the classifier. Option trees represent a useful middle ground between single trees and ensembles. At a fraction of the memory cost an option tree can provide comparable accuracy performance and superior prediction speed which are important factors in data stream processing. The results for pruning unnecessary structure were mixed. Indeed pruning may not be a viable option in a stream setting. Processor time is consumed and the potential accuracy improvements may not outweigh the cost. The idea of simply capping options, to the size of a single tree or to the memory limit of the processor is an attractive and effective low-cost alternative solution.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Database mining: A performance perspective. IEEE Transactions on Knowledge and Data Engineering 5(6), 914–925 (1993)
2. Ali, K.: Learning Probabilistic Relational Concept Descriptions. PhD thesis, University of California, Irvine (1996), `http://www.isle.org/~ali/phd/thesis.ps.Z`
3. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998)
4. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
5. Buntine, W.: Learning classification trees. In: Hand, D.J. (ed.) Artificial Intelligence frontiers in statistics, pp. 182–201. Chapman & Hall, London (1993)
6. Domingos, P., Hulten, G.: Mining high-speed data streams. Knowledge Discovery and Data Mining, 71–80 (2000)
7. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: KDD 2003. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 523–528. ACM Press, New York (2003)
8. Holmes, G., Kirkby, R., Pfahringer, B.: Stress-testing hoeffding trees. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, Springer, Heidelberg (2005)
9. Kohavi, R., Kunz, C.: Option decision trees with majority votes. In: Fisher, D. (ed.) Machine Learning. Proceedings of the Fourteenth International Conference, Morgan Kaufmann, San Francisco (1997)
10. Oza, N.C., Russell, S.: Online bagging and boosting. In: Artificial Intelligence and Statistics 2001, pp. 105–112. Morgan Kaufmann, San Francisco (2001)