The Effects of Noise on Efficient Incremental Induction (Extended Abstract)

Gerard V. Conroy (gvc@sna.co.umist.ac.uk)
David M. Dutton (ddutton@sna.co.umist.ac.uk)

Department of Computation,
UMIST,
PO BOX 88,
Manchester,
M60 1QD, UK.

Abstract

This paper presents an algorithm JITTER, that aims to eliminate any unnecessary work done whilst incrementally building decision trees. In particular, we illustrate how high levels of noise can greatly affect the efficiency of induction and how a straightforward approach can ameliorate these effects.

1 Introduction

Incremental learning has the potential to offer many advantages over one-step learning. New information can be added to a concept representation piecemeal, one can start with a small sample and refine the concept over time etc.. ID5 [2] achieves incrementality by maintaining statistics at each decision tree node that summarise class distributions, in terms of attribute values. ID5 can then judge whether a sub-tree has become sub-optimal with respect to entropy and will restructure a sub-tree in order to keep the most informative attributes near the top of the tree.

There are a number of aspects of incremental tree formation that can be reduced to increase the efficiency with which an algorithm works. These include entropy calculations, tree restructures, aborted restructures[1], the cost of individual restructures, and tree size/complexity. To reduce these quantities we propose the use of a more knowledge-based approach, one that utilises information easily available during induction. ID5 often does more tree restructuring than is necessary in order to maintain tree quality, e.g. accuracy (see [1]). Best

[1] Attempted restructures where entropy does not divulge a better test attribute thus leaving the tree unchanged.

'splits' are always tentative [3], especially during the early stages of a tree's development, as there is less information on which to base one's decisions. Entropy can cause a tree node to be swapped for another when in fact it does not increase the tree's accuracy or decrease its complexity.

2 JITTER

Our algorithm JITTER is based on ID5 and so we will concentrate on the extensions we have made. At each node in a tree is a list of the potential test attributes and their respective values, as for ID5. Each decision node has an associated weight that is used to indicate the efficacy (with respect to the partitioning of the example set) of the current test attribute. As an example is propagated down the tree, all that changes are the various counts at each node. The automatic entropy recalculations as seen in ID5 are omitted for the time being. When an example reaches a leaf, a correction factor (cf) is generated that is based on the class distribution of the examples at the leaf. The accuracy of a leaf should give a good indication of the effectiveness of the tests on the path to the leaf. A relatively pure leaf should thus give rise to increased confidence in it's contributory attributes while less pure leaves should result in a weakening. We use the ratio of majority class to the total of all classes at a leaf: *majority class count / total examples at leaf* and prepend this real with a '+' or '-' according to whether the last example to arrive is correctly or incorrectly classified, respectively. Once this cf is generated, it is weighted by the proportion of examples at that leaf to the total number in the tree. The factor is then propagated back up the tree from the leaf to the root and at each node along the path, we update the attached weight. Weights are updated using a formula that takes account of previous successes and failures and the current cf, similar to some neural net algorithms :- $w_i(t+1) = w_i(t) + cf$, where $w_i(t+1)$ is weight w_i at time $t+1$, similarly for $w_i(t)$ and cf is the correction factor. The weights along this 'active' path are now checked against zero, starting with the leaf and moving up to the root. If a weight is found to be less than or equal to zero then restructuring as per ID5.

In [1] we test JITTER with a number of training sets. In one set in particular, it was noted that as the percentage of noise was increased, the level of tree manipulation activity increased also. Specifically, the number of actual and aborted restructures increases significantly. In order to make use of this salient information, we add another weighting factor to the cf as it is generated at the leaf. This factor involves the number of global actual and aborted tree restructures and is calculated as: w = *no. aborted restructures / no.restructures.* It is used thus :-

$$IF\ no.\ aborted\ restructures\ >=\ no.\ restructures\ THEN\ cf\ =\ cf\ +\ w$$

Therefore as the number of aborted restructures increases in proportion to the number of actual restructures, we conclude that noise is playing a significant

part in the tree manipulation history and the node weights should be increased sufficiently to preclude further unnecessary change.

3 Evaluation

Each data set is randomly divided into a training set (two thirds) and a testing set (one third) and is repeated ten times. We use $I\hat{D}5$, instead of ID5, as it is more efficient with respect to entropy calculations and restructuring. The column headings signify time (seconds) to build the tree, the number of nodes and leaves (Nodes), the number of entropy calculations (Ecalcs), the number of restructures (Restrs), the number of *aborted* restructures (Aborts), the number of nodes restructured in total (Nodes Restr) and the accuracy of the tree (Acc) in percent. First we look at the seven-segment L.E.D. display. This concept has

Alg	Time	Nodes	Ecalcs	Restr	Aborts	Nodes Restr	Acc
ID3	8.6	158.0	236.0	0.0	0.0	0.0	72.7
$I\hat{D}5$	82.9	152.8	5411.2	127.8	890.9	1841.6	72.5
JITTER	14.5	161.4	462.1	7.4	11.1	46.1	72.5

Table 1: Results for 7-bit LED Set - 10% noise

1000 examples, ten classes, seven boolean attributes and is shown here with 10% noise. In this instance, accuracy and tree size are very similar but run-time, the number of entropy calculations and restructure costs are all significantly reduced for JITTER as compared to $I\hat{D}5$ (e.g. by some 82% for run-time). Indeed, JITTER's results are more comparable with the *non-incremental* ID3. Next we look at the L.E.D. set with 50% noise (table 2). The output, especially accuracy, deteriorates dramatically for all algorithms and illustrates that the ID3 family has a lot to improve upon. Nonetheless, JITTER again vastly improves in performance over $I\hat{D}5$. Our next set is the even parity concept with added noise

Alg	Time	Nodes	Ecalcs	Restr	Aborts	Nodes Restr	Acc
ID3	9.4	253.0	367.9	0.0	0.0	0.0	8.9
$I\hat{D}5$	1615.3	253.6	12974.8	506.5	1955.9	19261.0	8.8
JITTER	14.1	253.4	596.7	16.0	18.2	122.7	8.8

Table 2: Results for 7-bit LED Set - 50% noise

(10%). In this case we have 11 bits, 5 of which are relevant to classification (see table 3). In this instance JITTER has managed to improve over even ID3, with respect to node count and accuracy, albeit very slightly. The various costs of induction (entropy, restructures etc.) all show considerable differences between

Alg	Time	Nodes	Ecalcs	Restr	Aborts	Nodes Restr	Acc
ID3	37.9	1262.4	1987.3	0.0	0.0	0.0	62.4
$I\hat{D}5$	5967.8	1324.3	23090.2	594.5	1567.3	75986.7	64.8
JITTER	100.8	1255.4	6853.3	384.6	306.7	2615.6	64.8

Table 3: Results for Parity Error - 10% noise

$I\hat{D}5$ and JITTER. Finally, note with the 11-bit multiplexor with added noise (10%) JITTER again significantly decreases the time and effort taken to produce the trees over $I\hat{D}5$ and simultaneously improves node count and accuracy.

4 Conclusion

In [1] we show how ID4/5 can do unnecessary recalculations and restructures whilst building decision trees. Here, we show how simple extensions of these algorithms can use information available during the induction process to render them more knowledge-based. We use this information to implement an accuracy heuristic that reduces the automatically repetitive nature of the ancestor algorithms and significantly improves their inductive efficiency. This has been achieved with no significant decrease in accuracy. In addition, we have augmented our algorithm with a simple and effective measure for detecting and controlling the effects of noise during concept induction.

5 Acknowledgements

D. M. Dutton's work is funded by the E.P.S.R.C. (U.K.).

References

[1] Conroy, G. and Dutton, D. (1994), JITTER: A Lazy Machine's Guide to Induction, *Technical Report, UMIST-COM-AI-94-4* , Dept. of Computation, UMIST, PO BOX 88, Manchester, M60 1QD, UK.

[2] Utgoff, P. E., (1989), Incremental Induction of Decision Trees, *Machine Learning 4* , pp161-86, Kluwer Academic Publishers.

[3] Van de Velde, W., (1990), Incremental Induction of Topologically Minimal Trees, *Proceedings, 7th International Conference on Machine Learning* , pp 66-74, Morgan Kaufmann.