Decision DAGS – A new approach

Joost de Nijs

Table of Contents

Abstract

As computer science has developed as a field its focus has broadened from its core of designing and building systems and applications to applying the methods and ideas of the discipline to other areas. One such area is the classification problem. The classification problem deals with classifying elements of a set based on a set of attributes. This could range from deciding whether or not an incoming e-mail is spam to determining what type of clothing a given image represents. Machine learning has tackled this problem by seeking to predict future elements based upon the patterns the algorithms observe in a sample set. One of the primary methods of doing this is the Decision Tree. In this paper we seek to expand on the current techniques and algorithms in this subfield by introducing the idea of a Decision Graph. This approach seeks to shore up certain weaknesses inherent in the Tree through use of merging nodes. We demonstrate that this method has the ability to outperform traditional approaches and warrants further study.

Overview

Decision Trees

Decision Trees are based on the idea of creating a tree to represent the classification problem. In order to classify a new element the tree is traversed from top down. Each node splits on an attribute value of the data set, having one child for each potential value of that attribute. When a leaf is reached the new element is given the classification associated with that leaf. When creating a Decision Tree, the algorithm starts with the set of examples that have known classifications, and splits the examples into child nodes with each child having a certain attribute value fixed. There are a variety of methods of determining which attribute is chosen as the element to split on at a given node, but it generally involves the element which best separates disagreements in the classifications of the examples at the node. If a node has no disagreement in its remaining examples with regards to classification it doesn't subdivide any further. When a new item appears that needs to be classified, it traverses the tree based on its attribute values and then returns the classification of the leaf node at the end of the traversal.

Graphs

The method that this paper presents is substituting the Tree with a Directed Acyclic Graph (or DAG). We are not the first to take this approach, as several other implementations of this idea are already in wide circulation.[i] In particular, once a node has branched from its parent in the creation of a Tree it never references any other node and all further decisions to divide are independent of the rest of the Tree. Each node that isn't a leaf is in effect a root node of its own Decision Tree, entirely independent of the rest of the Tree. With the DAG we seek to exploit problems in which different parts of the tree have a similar structure and should in effect be merged.

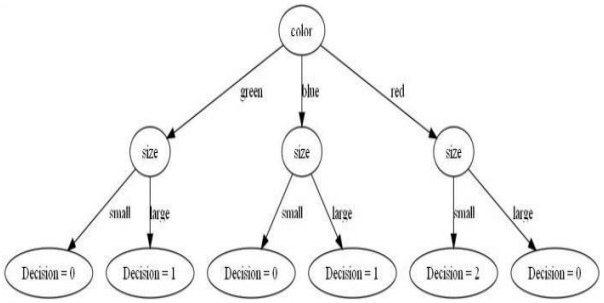

Figure 1: A Decision Tree that suffers from fracturing.

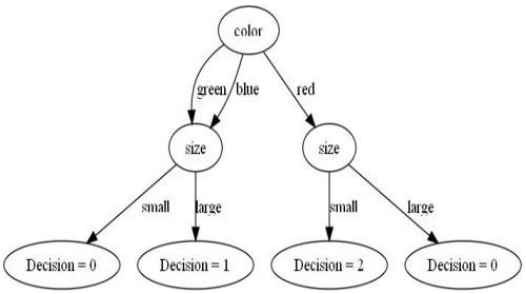

Figure 2: The equivalent Decision Graph depicting the same information but with the fracturing issue resolved.

One case where this would prove very useful is a DAG where the node is splitting on color (red, green, blue) but the real classification should be between (red, not red). In the Tree when this attribute splits there would be three separate children (as seen in Figures 1 and 2), while in the DAG the green and blue children would merge (providing twice as many examples to learn from). This case is often referred to in the literature as fracturing.[ii]

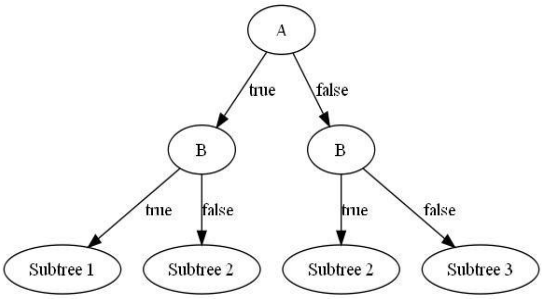

Figure 3: This is the tree presentation of the (A XOR B) situation. Note that Subtree 2 appears twice.

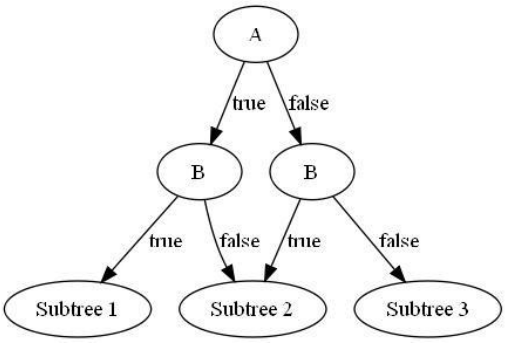

Figure 4: Here the DAG combines the two nodes that have the same structure, thereby better representing the underlying problem

Another case where the DAG would prove useful is a situation where there are two Boolean attributes, A and B, and a certain part of the tree is identical if (A XOR B). The repeated internal structure that this problem describes cannot be solved in the traditional tree structure (as demonstrated in figures 3 & 4). The DAG, meanwhile, would be able to combine the nodes A, ~B and B, ~A, in effect joining the sub-trees, and would then be able to learn more about their structure with fewer examples.

Implementation

Our implementation of the DAG was based on a top down approach. Our algorithm would begin by taking the full example set and placing all of those examples in the root node. The root node would then be placed in the fringe. The fringe is the set of nodes that still need to divide further. Nodes where all of the examples had the same classification would be regarded as a terminal leaf and not placed on the fringe. At each step of the algorithm a node would be removed from the fringe. This node would then be split based on the attribute that would result in the greatest decrease in overall entropy (or classification disagreement in the example set at a node) after taking into account the best merges for each child generated by the split of the active node. Once the attribute and subsequent ideal merges are identified, the planned split and mergers are executed. Any child nodes of the active attribute that did not merge and do not have a single common classification of their example sets are placed on the fringe. Child nodes that merged are already on the fringe as the node that they merged with is already on the fringe. This node is replaced by the merger of itself and the appropriate child node. Finally, any nodes that have uniform classification among its examples become terminal leaves in the DAG. The

exact process of splitting a node and looking for mergers is outlined below.

1. For each attribute class of the node complete the following steps:

    a. Split the examples at the node based on the given attribute.
    b. For each of the subsets, traverse the nodes of the fringe and check to see if the given subset could merge with the given node. If a merge is possible calculate the entropy of the resultant node. Maintain a list of the best merges sorted by the entropy delta.
    c. For each of the subsets, add the possibility of the node not merging with any nodes on the fringe to the list of best merges, inserted properly given the entropy of the node.
    d. Calculate the overall entropy delta by creating the best set of mergers of each subset, given the constraint that no two subsets can merge with the same node on the fringe.

2. Once each attribute has calculated its optimal entropy gain, choose the attribute to split on which has the greatest entropy delta.

3. Split the node into the appropriate child nodes and perform any mergers of the children with the respective fringe elements as detailed in step 1.

4. For each child of the given node, attempt to merge it with all the other children.

5. Add all child nodes where there is not yet a consensus on classification to the fringe.

6. Take the next node of the queue and go back to step 1. If there are no more elements in the queue at this depth, the fringe becomes the queue, and a new empty fringe is created. If the fringe is empty then the DAG has been constructed.

While the above algorithm defines the basic sequence that the algorithm follows in its execution, it doesn't fully detail the most critical element of our method, the merger. A merge consists of two unique steps in the algorithm. First, the merge function strips both sets of all attributes which the other has already split on. Then the two sets are evaluated to insure that there are no instances in which the same set of unassigned attributes cause a disagreement in classification. Finally, a tally is taken of the number of instances of overlap in identical set of attribute values leading to the same classification is taken, and if that tally is high enough (we tested a number of different merge threshold values in our experiment); the two data sets are considered merge-able.

The set of potential mergers are then ranked based on their entropy delta. The entropy delta is a calculation which involves comparing the total entropy of the potential merged node, to the sum of the entropies for the two nodes separately. In general, the goal is to minimize the total entropy of the fringe, so instead of comparing the entropy of the merged node to the entropy of just the child node, the comparison takes into account the total change in entropy of the entire fringe by completing the merger. For a given node n, containing S examples to be classified on attribute A, where p_i is the number of examples where attribute A = i, we calculate the entropy with the following equation:

$$Entropy(S) = \sum_{i=1}^{c} -p_i * \log_2 p_i$$

This is a commonly used formula in machine learning algorithms as it seeks to reduce the length of the amount of data needed to represent the model describing the remaining example set. This builds off of the Minimum Message Length theory of information theory as presented by Wallace and Oliver.[iii]

Once each potential child of the node has its ordered lists of best merges, the algorithm then chooses the best combination of choices from each child, with the constraint that no two children should merge with the same node. In future implementations of the DAG multiple children should be allowed to merge with the same node, but the current structure of the code made the implementation of this feature quite difficult. The code seeks to remedy this shortcoming with the potential mergers of child nodes after they have been created (and merged with the recommended nodes for each child), but this solution can be improved upon in future work.

Previous Work in DAGs

The idea of using a graph instead of a tree for use in machine learning problems is not new. The roots for this approach to machine learning initially emerged from the idea of a Decision Forest. Decision Forests are Decision Trees that deal with the fracturing problem, and were first introduced by Volesu and Uther.[iv] In their work they demonstrate that merging several nodes with a common parent into a single node can improve model accuracy. This model fell short of dealing with the repeated internal structure problem. This was addressed with the introduction of decision graphs into the field.[v] The first forays into the field showed promise, with improvements to accuracy. However, massive increases in run-time for the model to be created from the learning set were evident even then.

For most research on Decision Graphs the algorithmic approach consisted of three core steps. First the algorithm would scan through all open nodes (those without children, and without a homogenous classification), and calculate the best attribute split for each one as if it were a Decision Tree. From all these nodes it would record the node/split combo which reduced the entropy of the system the most. Next, the algorithm would scan through all combinations of open nodes and calculate a merger of the nodes. A delta entropy would be calculated based on the difference of entropy between the merged set, and the individual nodes. Finally, the algorithm would choose the move, either a merge or a split, which caused the greatest decrease in entropy for the system as a whole. This process would be repeated until no open nodes remained. More recent work has focused on streamlining multi-way joins[vi], and dealing with dynamic approaches to prevent premature merges[vii].

The primary distinction that differentiates our approach to previous work in this field lies in the manner of designating the next move made in learning the tree. In traditional approaches all possible merges and splits are examined before deciding to make the move that results in the best decrease in entropy. In an ideal world our algorithm would also seek to check all open nodes and calculate the ideal move on a universal level. However our algorithm decides the split by also taking into account for the merges of the children against all possible mergers, the computational complexity of checking all open

nodes would spike far beyond reasonable levels. With the implementation as is (where the node to split on is randomly selected), the total run time still surpassed a month worth of CPU cycles. In fact, the current implementation was so computationally heavy that the largest training sets could not be run in a reasonable time.

Experimental Design

In order to test our new algorithm, we needed to construct a test sweep that evaluated our method against the traditional approaches. Additionally, we needed to use a wide variety of different data sets to test the algorithms on.

Name	Elements	Learning Set Size	Test Set Size	Classification Groups	Number of Attributes	Average Attribute Choices	Max Attribute Choices
Balance-Scale	625	{50, 100, 150}	475	3	4	5	5
Cars	1728	{50, 100, 355}	1473	4	6	3.5	4
Mushroom	8124	{10, 25, 50, 100, 250, 500, 1000}	7124	2	22	5.7	10
Nursery	12960	{50, 100, 250, 500, 1000}	11960	5	8	3.4	5
Connect 4	67557	{100, 250, 500, 1000, 2500, 5000}	62557	3	42	3	3
Poker	1025010	{500, 1000, 2500, 5000}	1000000	10	10	8.5	13

Table 1: Datasets used in the Experiment, with essential properties of the datasets outlined

Data Sets

The tests used six different databases from the University of California at Irvine repository.[viii] In particular we used the mushroom dataset, the poker dataset, the nursery school dataset, the balance scale dataset, the car rating dataset, and the connect 4 dataset. Each of these datasets has only Categorical Attribute fields, and can have between 2 and 10 unique classifications of dataset elements. Additionally, all of the datasets used did not have any misclassified entries. Finally, while the poker dataset did have a pre-divided train set and test set, for the rest of datasets there was only one large dataset. Furthermore, the poker set was not cross-validated due to the massive computation time it required, meaning that the results are not as strong as for the other datasets. In order to divide the data set the corpus was divided into two unique parts of predetermined size. Each of these splits was conducted ten times to insure that the results of the experiment were not an artifact of a particular split, but instead indicative of the structure of the data set and the algorithms used to evaluate them.

Pruning

Although the basic Decision Tree forms a good baseline in terms of accuracy, the field has a number of additional techniques on top of the basic Tree. Our experimental design tests against two of these methods. The first of these is pruning. While the first steps of the typical decision tree involve legitimate separations of the learning set into distinct classifications, it is possible to take this too far by splitting the example set too many times. This error is especially common in data sets where there is a

degree of misclassification as just one misclassified example can lead to unnecessary splitting of nodes and cause a decrease in evaluation accuracy. In pruned trees, the initial learning data set is split into two components, the base set which builds a tree as normal, and a validation set[ixx]. The tree is then pruned by removing different leaf nodes from the tree so long as such action doesn't negatively impact the classification accuracy of validation set. There are a number of ways to prune, including top-down and bottom-up approaches.

Bagging

The other advanced technique we used was bagging. Bagging involves taking a random subset of the learning set and creating a Decision Tree (or Pruned Decision Tree) from it, and then repeating the process a number of times (usually 10-20). The subsets are typically a sizeable fraction (usually half) of the learning set, and separate subsets can contain the same element. When classifying a new item the algorithm lets each Tree it created classify the item, and then takes the majority vote and returns that value as the item's classification. Since bagging does not affect the underlying algorithm, we applied this technique to the traditional Tree, the Pruned Tree, and also to our DAG.

Experimental Procedure

A single test sweep consisted of the given learning set being used to create a Decision Tree, 10 Pruned Trees, 10 Bagged Trees, 10 Bagged Pruned Trees, and 10 DAGs and Bagged DAGs for each merge threshold value being tested (For all test sweeps the values 1, 2, 3, 4 and 5 were tested). Each of these Trees or DAGs would then be scored for accuracy against the test set, with a percentage of correct identifications recorded. For learning set sizes that were smaller than the size of the learning corpus, a random subset of the appropriate size would be selected from the learning set and used for all test runs. The reason for using only one Decision Tree is the deterministic nature in which the Decision Tree is created—for all the other methods there is a degree of randomness involved, thereby requiring a larger sample size to give a more accurate depiction of the methods performance. For those data sets that we split into test and learn sets ourselves, the full test sweep for each learn set size was run for all the separate splits.

Results

After running each of the Data Sets through ten iterations of the test sweep, the accuracy rates for the various methods was aggregated and the results are displayed in Table 2. In the table, each cell has an average accuracy as well as a margin of error equal to standard deviation for the data. When looking at the results it is important to note that the first three columns of data correspond to non-bagged methods, and the second three correspond to those same methods being bagged. With regard to the non-bagged data sets, the margins of error are generally too high to prove statistical significance, though it should be noted that apart from the poker data set, the DAG was always within the margin of error of the traditional methods. Additionally, although there are no instances of the DAG being better with regards to statistical significance it definitively outperforms the completion on the balance and nursery data sets.

		Normal			Bagged		
DataSet	Train Set	Tree	Pruned	DAG	Tree	Pruned	DAG
balance	150	64.0 ± 1.5	63.2 ± 2.6	68.8 ± 4.6	69.3 ± 3.1	70.4 ± 2.6	75.4 ± 2.8
cars	355	85.1 ± 1.2	83.5 ± 1.6	84.7 ± 2.7	84.7 ± 1.6	83.5 ± 1.2	86.1 ± 1.6
connect 4	5000	66.3 ± 1.2	65.6 ± 1.2	66.0 ± .9	73.8 ± .7	73.8 ± .6	73.2 ± .5
mushrooms	1000	99.7 ± .1	99.6 ± .2	99.5 ± .2	99.7 ± .2	99.6 ± .2	99.6 ± .2
nursery	1000	89.9 ± .7	89.0 ± .8	91.1 ± 1.9	90.3 ± .7	89.9 ± .7	92.2 ± .8
poker	5000	53.2	52.2 ± .4	47.6	57.8 ± .2	56.9 ± .2	49.1 ± .1

Table 2: Experimental Results for Largest Train Sets for each Data Set

Switching focus to comparing the results of bagged methods and their performance, trends similar to those observed in the non-bagged results are noted. As before, the poker set presented a large decrease in performance, while the other areas saw much more promising results. In particular the DAG performed very well in the balance and cars set, just missing statistical significance in the former, and in fact vastly outperformed the other methods in the nursery data set, to a level outside the margin of error. Across the board, with the training set sizes observed, there was a noticeable improvement in testing accuracy when comparing the non-bagged and bagged DAGS. However, there were instances (which involved much smaller training sets) in which the bagged methods performed noticeably worse across the board.

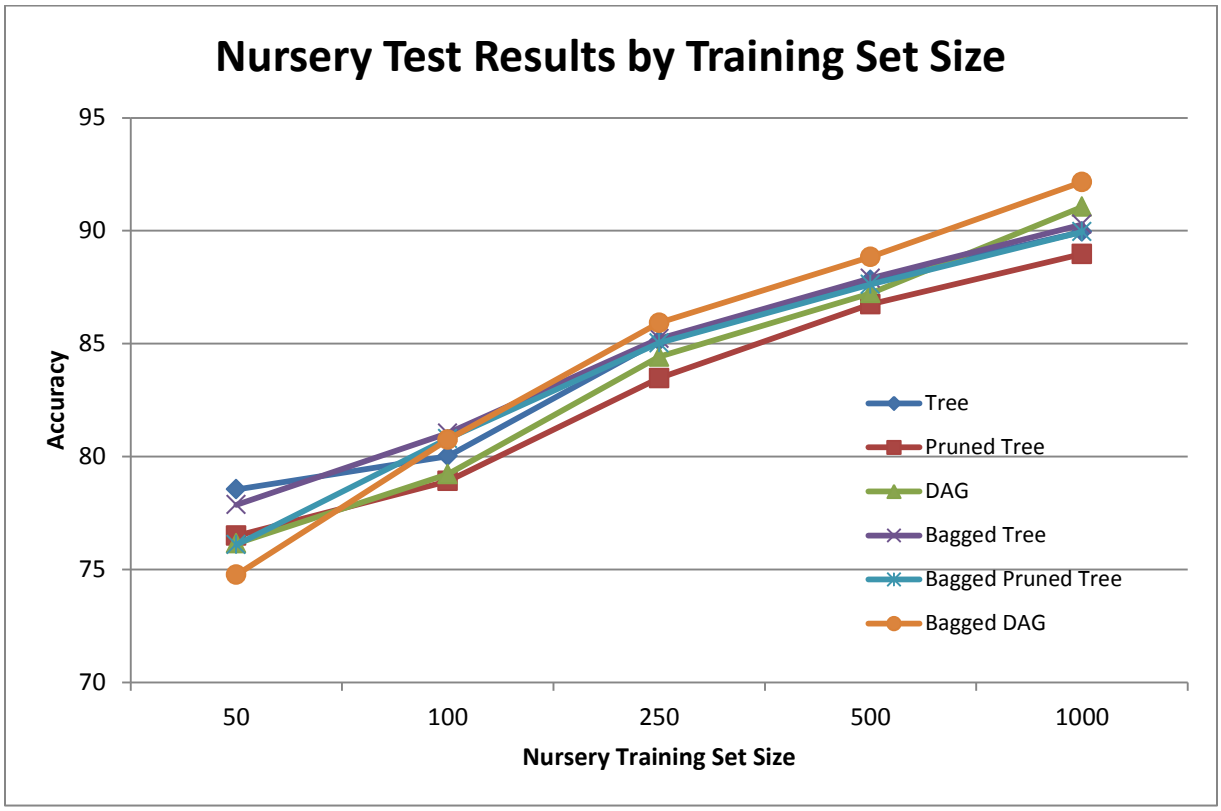

Figure 1: This chart represents the accuracy of the 6 methods for different training set sizes on the Nursery Data Set

The case of the nursery data set is rather interesting, as the highest two scores were the bagged DAG and normal DAG respectively. As such, this data set deserves an extra level of analysis. In particular Figure 1 tracks the performance of the six different methods with respect to the size of the training set. The most interesting trends are the rise of the bagged DAG and DAG from the two worst methods at a training set size of 50 to their rise to the head of the pack once the training set reached 1000. Because our method is always looking to merge and the thresh-holds for doing so are rather low, early training sets experience a lot of false merging which negatively impacts their performance with respect to their non-merging competitors. However, the chance of false mergers decreases as the training set increases in size. In particular, the merging portion of the algorithm can more accurately identify the commonalities between different open nodes and subsequently more accurately predict the existence of a repeated structure in the sub-graphs. As such there is a dramatic increase in performance with the bagged DAG eliminating almost 70% of its misclassifications between the smallest and largest training set sizes compared to only a 53% reduction for the Decision Tree.

The other case that warrants additional focus is the poker Data Set. In theory a Data Set that is almost entirely case-studies of repeated structures and fracturing should be the ideal Data Set for an algorithm like the DAG. However, the training set was far to sparse with respect to all possible attribute combinations. This created a situation in which the algorithm had far too little information to properly pinpoint nodes that should be merged. This sparseness led to the algorithm deciding to merge nodes that shouldn't be merged, given full knowledge of the problem. These mis-mergers in turn created a representation of the data set that is far removed from the actual structure of the problem, which in turn decreased classification accuracy on the test set.

Future Work

There are two main areas in which future research can seek to improve the performance of our DAG algorithm. The simpler of these two metrics involves changing the thresh-hold parameters for merging in the model. In particular, relaxing the thresh-holds to allow for false classification, and using a metric that is not a simple integer threshold but instead one that is dynamically adjusted based upon the projected size of the node's theoretical example space could provide useful improvements. While, the idea of using a percentage-based thresh-hold was experimented with early in the design of the model, it was opted against in an effort to limit the scope of investigation, and due to the integer thresh-hold yielding better results on early test sweeps.

The other area that is ripe with possibility for improvement lies in bringing our method more in line with the established methodology of the Decision Graph field. In particular, the lack of a global view when selecting the next move, with the focus instead resting on the potential moves of a randomly selected node lead to less than optimal results. However, our method demonstrated gains over the traditional methods of Trees and Pruned Trees similar to early work on Graphs. Therefore a merging of the two different approaches could lead to even better results. The main downside to any research in this area lies in the added layer of computational complexity that this adds to the equation. By performing the calculations required for each step in computation for all open nodes, the complexity increases in magnitude by the number of open nodes at any point in time. Therefore, improving the

data structures to optimize the number of re-calculations to only dealing with changed nodes is necessary before any larger problems can be tackled.

Conclusion

Overall, the idea of a Decision DAG is a promising one, though a computationally intensive one. Our results demonstrate the power of this algorithm in certain types of problems, specifically those in which the underlying data set has repeated structure or fragmentation issues that our algorithm can exploit. While our approach took a different take on the DAG by combining the split and merge into a single step, as compared to the single stepped approach dominant in the field, we still demonstrated improvements over the traditional approaches. This leaves open the avenue for a combined approach of global view with multi-step considerations at each level.

---

[i] OLIVER, J.J., D.L. DOWE and C.S. WALLACE, 1992. Inferring decision graphs using the minimum message length principle. *Proc. 5th Joint Conf. Artificial Intelligence.*

[ii] William T. B. Uther and Manuela M. Veloso. 2000. The lumberjack algorithm for learning linked decision forests. In *Proceedings of the 6th Pacific Rim international conference on Artificial intelligence* (PRICAI'00), Riichiro Mizoguchi and John Slaney (Eds.). Springer-Verlag, Berlin, Heidelberg, 156-166.

[iii] OLIVER, J.J., D.L. DOWE and C.S. WALLACE, 1992. Inferring decision graphs using the minimum message length principle. *Proc. 5th Joint Conf. Artificial Intelligence.*

[iv] William T. B. Uther and Manuela M. Veloso. 2000. The lumberjack algorithm for learning linked decision forests. In *Proceedings of the 6th Pacific Rim international conference on Artificial intelligence* (PRICAI'00), Riichiro Mizoguchi and John Slaney (Eds.). Springer-Verlag, Berlin, Heidelberg, 156-166.

[v] J.J. Oliver. Decision Grpahs- An Extension of Decision Trees. *Proc. 4th International Workshop on Articifical Intelligence and Statistics*, pages 343-350, 1993.

[vi] P.J. Tan and D.L. Dowe. MML inference of decision graphs with multi-way joins. *Proc. 15th Australian Join Conference on AI, LN AI 2557 (Springer)*, pages 131-142, Canberra, Australia, 2-6 Dec. 2002.

[vii] Peter J. Tan and David L. Dowe. 2002. MML Inference of Decision Graphs with Multi-way Joins. In *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence* (AI '02), Bob McKay and John K. Slaney (Eds.). Springer-Verlag, London, UK, UK, 131-142.

[viii] Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[ix] Michael J. Kearns and Yishay Mansour. 1998. A Fast, Bottom-Up Decision Tree Pruning Algorithm with Near-Optimal Generalization. In *Proceedings of the Fifteenth International Conference on Machine Learning* (ICML '98), Jude W. Shavlik (Ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 269-277.

[x] Tapio Elomaa and Matti Kaariainen. 2001. An analysis of reduced error pruning. *J. Artif. Int. Res.* 15, 1 (September 2001), 163-187.