Inference of Decision Graphs using the Minimum Message Length Principle

Tobias Witt[⋆]

Data Mining Algorithms
Chair for Bioinformatics and Information Mining
Department of Computer and Information Science
University of Konstanz, Germany

Abstract. Decision graphs are an extension of decision trees originally proposed by Oliver, Dowe and Wallace (1992). This paper shows how decision graphs can be inferred from a set of training examples using the *Minimum Message Length* Principle (Wallace and Boulton, 1968). It furthermore presents the implementation of the decision graph algorithm for the data analytics platform KNIME (Berthold et al., 2007). Empirical tests show that the decision graph classifier is a serious alternative to standard tree classifiers.

1 Introduction

This paper presents the implementation of an algorithm for *supervised concept learning*. Given a set of training examples, the goal of supervised learning is to find hypotheses which approximate some target function on the whole instance space. More specific, the procedures described below use training data to learn a function that determines the *class* of instances by considering some or all of their observed characteristics. Within a specific family of functions, the aim is to find the function that permits the highest probability of correctly predicting the class of new instances (Oliver, 1993). The variables describing characteristics of the instances are referred to as *attributes*. The data consists of vectors with the values of the attributes and the class for each instance. The training data is a sample from the whole instance space.

A commonly used representation for the learned function is a *Decision Tree*. An example of such a tree is given in Figure 1 for a binary class variable. The

[⋆] `tobias.witt@uni-konstanz.de`

class value is `true` if the expression $(A \wedge B) \vee (C \wedge D)$ holds and `false` otherwise. The tree consists of *decision nodes* and *leaves*. In each decision node the remaining data is partitioned according to the distinct values of the attribute used for taking the subsets. Each path from the root to a leaf represents one *decision rule* determining the class of instances based on specific constraints on the attributes. The attribute constraints partition the set of instances into disjoint *categories* which correspond to the leaf nodes in the tree (Oliver, 1993). The aim is to construct trees in which the observations ending up in the same category predominantly belong to one class (Quinlan and Rivest, 1989). Classification of new objects is done by traversing the tree according to the object's attribute values until a leaf node is reached. The prediction for the instance is the default (usually the most frequent) class in this leaf.

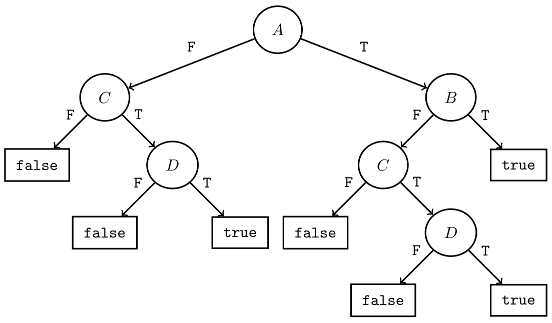


Fig. 1: Decision tree representation of $(A \wedge B) \vee (C \wedge D)$

For each class it is possible to construct a decision rule with the corresponding class value as the consequent. This is done by combining all paths in the tree leading to a leaf with the respective class value as default. Since each path from the root to a leaf is a conjunction of attribute constraints, the resulting rule is a disjunction of conjunctions, i.e., the rule is in *disjunctive normal form* (DNF). For example, we can construct the following rule from the tree in Figure 1:[1]

$$(\neg A \wedge C \wedge D) \vee (A \wedge \neg B \wedge C \wedge D) \vee (A \wedge B)$$

[1] Since the class variable and all attributes are boolean it is sufficient to express the rules as logical expressions. For instance, one could also write: IF $A =$ `true` AND $B =$ `true` THEN class `true`.

Using boolean algebra, this expression can be simplified to $(A \wedge B) \vee (C \wedge D)$. In general, all logical formulas can be converted to an equivalent DNF. Thus, a Decision Tree can depict arbitrary propositional rules.

Despite the general applicability of decision trees, the tree representation suffers from two major shortcomings. The first one is known as the *replication problem*. To illustrate this problem, consider again the decision tree in Figure 1. Notice that in order to represent the underlying proposition, the subtree for $(C \wedge D)$ needs to be duplicated in the tree. In general, the decision tree representation tends to be inefficient for disjunctive propositions (Pagallo and Haussler, 1990). The consequence of this inefficiency is that more data is needed to properly learn to underlying concept (Oliver, Dowe and Wallace, 1992; Tan and Dowe, 2002). As Oliver (1993) asserts for the example in Figure 1, if N instances are required to learn the subterm $(C \wedge D)$, the same algorithm will need at least $2N$ instances to learn the complete expression $(A \wedge B) \vee (C \wedge D)$.

The second shortcoming is known as the *fragmentation problem* and arises when the data contains high arity attributes. If these attributes are used to split decision nodes, the data will be quickly partitioned into many small subsets. This characteristic of decision trees again entails that more data is needed to correctly learn the underlying function (Oliver, Dowe and Wallace, 1992; Tan and Dowe, 2002).

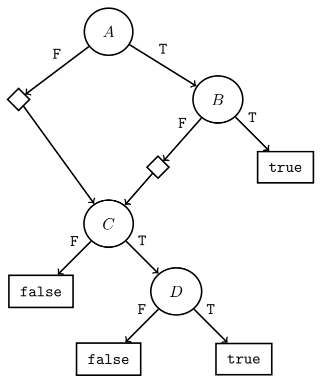


Fig. 2: Decision graph representation of $(A \wedge B) \vee (C \wedge D)$

An elegant way to avoid both the replication and the fragmentation problem is to use a *decision graph* (directed acyclic graph) instead of a tree (see, Oliver, Dowe and Wallace, 1992; Oliver, 1993; Tan and Dowe, 2002, 2003). In difference to trees, the graph representation allows two or more nodes to have a common child. The operation of creating this common child by concatenating the instances in the single nodes is called a *join*. Decision graphs and trees can represent the same set of decision rules, however, the graph representation is more efficient for disjunctive concepts (Oliver, 1993). Figure 2 shows the decision graph for $(A \wedge B) \vee (C \wedge D)$. The graph is easier to interpret and no duplication of subtrees is needed. This superior representation is obtained by using the join operator to create a new node containing those observations for which $\neg(A \wedge B)$ holds.

The construction of decision graphs with training data is similar to the procedure for decision trees. However, instead of just deciding which attribute to use for a split at a specific node, it is furthermore necessary to test if a join of two or more nodes can improve the graph. Therefore, a criterion is needed to decide if a split, a join, or no alteration at all is most appropriate. One such criterion is given by the *Minimum Message Length* (MML) principle (Wallace and Boulton, 1968; Wallace, 2005). The next section introduces the MML principle and describes how it can be used to infer decision trees and graphs from a set of training instances. Afterwards the decision graph algorithm and its implementation for KNIME (Berthold et al., 2007) are presented. Its performance in terms of prediction accuracy is then compared to a standard decision tree algorithm.

2 Minimum Message Length Principle

2.1 Induction Principle

The MML principle provides a general guideline to infer the best model from a set of training examples (Oliver, 1993). Its basic idea can be motivated using a communication problem involving a sender and a receiver (see Quinlan and Rivest, 1989). Consider the following scenario: Person A owns a data set with N instances, each belonging to exactly one of several classes. The data also includes a number of attributes describing the instances. Person B owns the same data table except that the class column is missing. Person A wants to send Person B an exact description of the missing class column using *as few bits as possible*. Basically two options are available: First, Person A could directly transmit the class column, requiring the transmission of N bits. Second, Person A could transmit a model (e.g., a set of rules) which Person B can use to deduce the missing class values from the known attributes. This strategy might require much less bits if the class variable depends to a significant extent on the attributes (Quinlan and Rivest, 1989).

The MML principle states that, given a set of data, the best model is the one which requires the fewest possible bits to describe the data, that is, the model allowing for the shortest *message* Person A can send to Person B in order to transmit the missing class column (Wallace and Boulton, 1968; Wallace, 2005). The message is a binary sequence consisting of two parts: (1) a description of the hypothesis $h \in H$, where H is the universe of eligible hypotheses, and (2) the data D explained with the help of h (Quinlan and Rivest, 1989; Wallace and Patrick, 1993). In order to transmit information using such a message, the sender and receiver must agree on a code to represent information in binary form. Following principles of classical information theory, the length of the full message - using an optimal code - is given by:

$$\text{length}(h, D) = -\log_2(P(h, D)) = -\log_2\left(P(h)P(D|h)\right)$$

Applying Bayes' rule reveals that searching for the hypothesis allowing the shortest message is the same as searching for the hypothesis exhibiting the highest probability given the data:

$$\begin{aligned}\arg\max_{h\in H} P(h|D) &= \arg\max_{h\in H} \frac{P(h)P(D|h)}{P(D)} \\ &= \arg\max_{h\in H} P(h)P(D|h) \quad \text{(since } P(D) \text{ is a constant)} \\ &= \arg\min_{h\in H} -\log_2\left(P(h)P(D|h)\right)\end{aligned}$$

Notice that the length of the full message is equal to the sum of its parts:

$$\begin{aligned}\text{length}(h, D) &= -\log_2\left(P(h)P(D|h)\right) \\ &= -\log_2(P(h)) - \log_2(P(D|h)) \\ &= \text{length}(h) + \text{length}(D|h)\end{aligned}$$

Now, it is easy to see that there is a trade-off between the complexity of the hypothesis and the complexity of the data given the hypothesis. For complex models, less bits are needed for the second part, but more bits to transmit the model in the first part of the message. In this way, the MML principle prevents overfitting by design.

2.2 MML for Decision Trees

The MML principle has been successfully applied to construct decision trees (see, e.g., Wallace and Patrick, 1993) In order to transmit the class of instances using a decision tree, the message must contain two parts: In the first part the complete structure of the tree is depicted (*structure message*). That is, all information the receiver - who only observes the attributes - needs to reconstruct the tree.[2] Let S be the universe of all eligible tree structures. The number of bits needed to transmit a specific structure $s \in S$ is given by $-\log(P(s))$. Wallace and Patrick (1993) propose a procedure to compute the probability $P(s)$. Refer to their paper for a detailed description of the applied coding technique.

In the second part of the message the distribution of class values within each leaf node must be transmitted (*category message*). Given M distinct classes, Wallace and Patrick (1993) assume a generalized symmetric Beta prior over the unknown class probabilities p_m $(m = 1, 2, \ldots, M)$ in each leaf category:

$$f(p_1, \ldots, p_m) = \prod_{i=1}^{M} p_i^{(\alpha-1)} \quad \text{with} \sum_{i=1}^{M} p_i = 1,\ p_i > 0$$

For $\alpha = 1$, this gives a uniform prior distribution. For values of α less than one, greater weight is placed on rather pure distributions, that is, those distributions in which only one or very few classes exhibit probabilites considerably different from zero. In the extreme case $(\alpha = 0)$ all weight is placed on the M possible single class distributions. As Wallace and Patrick (1993) note, a useful tree should produce leaf categories each predominantly of a single class. This expectation is not expressed by the uniform prior. Therefore, values of α smaller than one should be more appropriate in general.

To encode each leaf category, Wallace and Patrick (1993) suggest to use an *incremental code.* Each instance within a leaf category is encoded with length $\log_2(q)$, where q is the updated (expected) probability of the element's class after having observed the previous elements. More specific, if among the first j instances within a leaf category, i_m have had class m, the class of the $(j+1)$th instance is encoded assigning a probability

$$q_m = \frac{i_m + \alpha}{j + M\alpha}$$

[2] This includes the type of all nodes (decision node or leaf), the edges connecting them, and the attributes used for splitting in each decision node. If a split is performed on a continuous attribute, the cut value must be specified as well.

to class m. Thus, q_m increases with the number of instances of class m already observed, so that less bits are needed to transmit further instances of the same class. Therefore, the sum of the category messages will be shorter for trees with rather pure leaf categories. For more details on the incremental coding technique, see Wallace (2005).

2.3 MML for Decision Graphs

The coding scheme proposed by Wallace and Patrick (1993) can be applied to decision graphs by decomposing the graph into a sequence of decision trees (see, Tan and Dowe, 2002, 2003). This procedure is illustrated in Figure 3 for the decision graph from Figure 2. By treating all join nodes as leaves, the separate decision trees can be directly encoded with the scheme presented above. In addition to the bits needed for each tree, the *joining pattern* - an instruction on how to combine the separate trees - must be transmitted. A detailed description on how to calculate the number of bits to communicate a specific pattern can be found in Tan and Dowe (2002). The complete message for a decision graph consists of the structure and category messages of all individual trees plus the joining pattern messages.

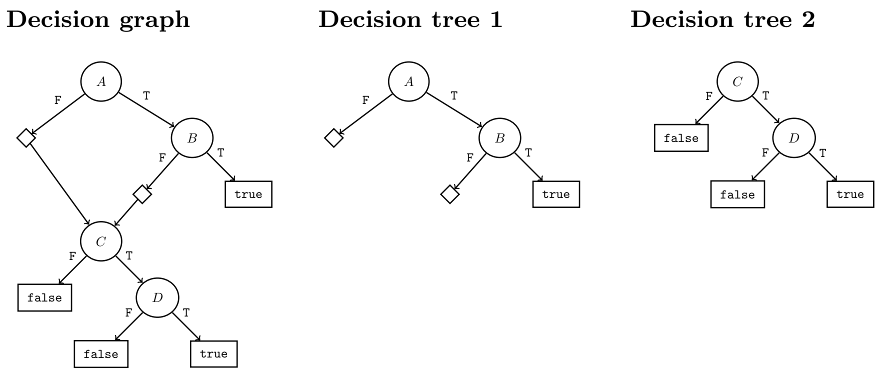


Fig. 3: Decomposing a decision graph into two decision trees.

3 Decision Graph Algorithm

3.1 Growing Decision Graphs

Algorithms to construct decision graphs given a set of training instances were presented by Oliver, Dowe and Wallace (1992), Oliver (1993), and Tan and Dowe (2002, 2003). Starting with a single leaf node as the root, Tan and Dowe (2002, p. 137-138) propose to iteratively repeat the following growing procedure until no further improvement can be achieved:

1. For each leaf L, perform tentative splits on each available attribute in the leaf, and determine the attribute A that will lead to the shortest message length when L is split on A.
2. For each leaf L, perform tentative joins with other leaves.
3. Sort the alterations from step 1 and step 2 by their communication savings. Choose the alteration (from step 1 or 2) that has greatest saving.

In step 2, only joins involving leaves with communication savings when splitting on an identical attribute are considered since only those can form a join that has a possible message length saving (Tan and Dowe, 2002). In most cases, this tremendously reduces the number of tentative joins that have to be performed in each iteration. Notice further, that nodes resulting from a join operation in the current iteration cannot be involved in a join in the following iteration.

In an attempt to replicate this algorithm, it was discovered that join operations were performed rarely. If at all, joins were executed in deep levels of the tree when no split further reduced the message length. To induce more joins in upper levels, step three of the procedure above can be altered Instead of performing the operation with the highest message length savings in each iteration, one might always prefer the join operation with the largest savings (if greater than zero) to the best split operation. The implementation of the algorithm presented in the next subsection allows for this altered growing procedure. It is also used for the empirical tests in Section 4.

3.2 Implementation in KNIME

The decision graph algorithm was implemented in Java for the usage in KNIME. Figure 4 shows the dialog box of the node and a simple example workflow. The decision graph node has two input ports. The first port is for the training data. To construct the decision graph, all columns of the training data (except of the class column) are utilized. The node accepts both categorical and numerical attributes. Numerical variables are handled by searching for the binary split producing the largest savings in message length. The second port takes the data used for prediction. Note that the training and test data must have identical columns. The node returns the test data with an additional column containing the class prediction. As shown in the example workflow, the output can be used to compare the predictions with the actual class values using the `Scorer` node.

In the dialog box the following options can be specified:

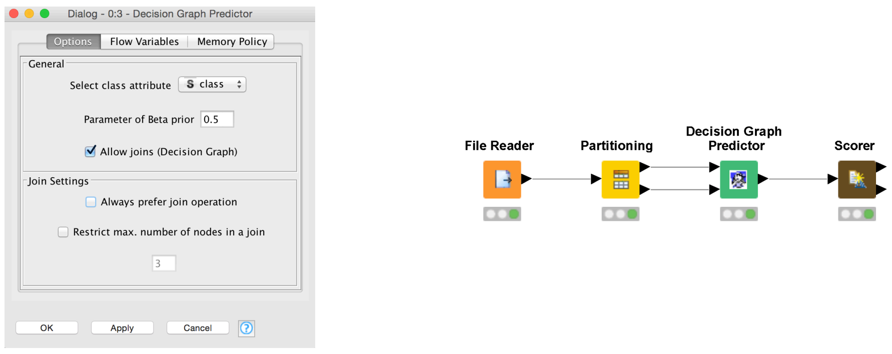
class attribute The column containing the class of instances. This must be a nominal variable. For string variables with too many distinct values the execution may fail when no domain information is available.

α-parameter of Beta prior The single parameter of the generalized symmetric Beta prior distribution as described in Section 2.2. This must be a number between 0 and 1.

allow joins If this option is not checked, the node learner will construct an ordinary decision tree using the MML principle to select the splitting attributes.

prefer joins If this option is checked, the join operation with the largest savings in message length (if savings are larger than zero) will always be preferred to the best split operation. If unchecked, the operation with the highest communication savings (split or join) will be performed in each iteration.

max. nr. of nodes in join This option allows to restrict the maximal number of nodes possibly involved in one join. For example, it can be used to allow only binary joins.



Dialog - 0:3 - Decision Graph Predictor

Options | Flow Variables | Memory Policy

General

Select class attribute: S class

Parameter of Beta prior: 0.5

[✓] Allow joins (Decision Graph)

Join Settings

[ ] Always prefer join operation

[ ] Restrict max. number of nodes in a join

3

Fig. 4: The decision graph node in KNIME (dialog box and example workflow).

4 Empirical Tests

4.1 MONK's 3rd Problem Data Set

The 3rd MONK's data set from the UCI machine learning repository[3] has 432 observations. It consists of a binary class variable and six nominal attributes ($A_1 - A_6$) with up to four distinct values. The data was constructed based on the following target concept:

$$\text{IF } (A_5 = 3 \wedge A_4 = 1) \vee (A_5 \neq 4 \wedge A_2 \neq 3) \text{ THEN class } 1$$

To compare the decision graph classifier to the decision tree algorithm, 10 independent 10-fold cross-validations were executed on the full data set. For comparison, the KNIME decision tree learner with *Minimum Description Length* (MDL) pruning was used. For more information on the MDL-based pruning technique, see Mehta, Rissanen and Agrawal (1995). For a comparison between MML and MDL, see Wallace and Dowe (1999).

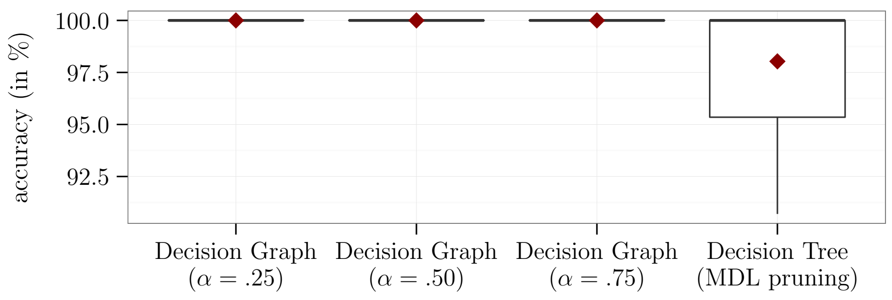


Fig. 5: Results of 10-fold cross-validation repeated 10 times. The mean accuracy is shown as a red dot.

The graphic in Figure 5 shows boxplots for the obtained classification accuracies. As can be seen, the decision graph algorithm always reconstructed the original target function (100% mean accuracy). In contrast, the decision tree algorithm misclassified several instances in a couple of iterations (98% mean accuracy). In case the decision tree classifier reproduced the target function, it constructed the

[3] `http://archive.ics.uci.edu/ml`

tree depicted in Figure 6. The corresponding decision graph constructed by the decision graph classifier is shown in Figure 7. It gives a superior representation of the underlying concept.

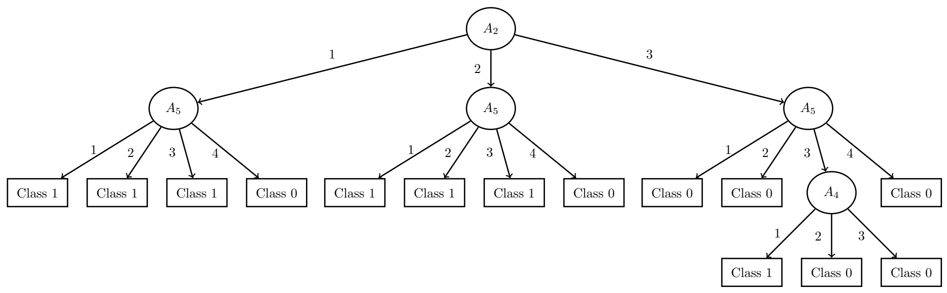


Fig. 6: Decision Tree for MONK's 3rd problem data set.

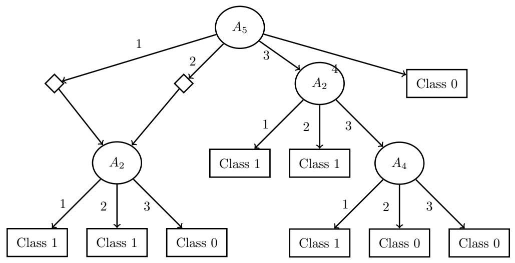


Fig. 7: Decision Graph for MONK's 3rd problem data set.

4.2 Accuracy Tests

Further accuracy tests were performed on frequently used data sets from the UCI Machine Learning Repository. Again, for each data set and each classifier a 10-fold cross-validation procedure was repeated 10 times. The results are summarized in Figure 8. For the `abalone` and `pima` data sets, the variants of the decision graph algorithm produce nearly the same results as the decision tree algorithm with MDL pruning implemented in KNIME. In fact, the graph learners do not perform any join operations in these two cases. For several other data sets, the decision graph make (on average) slightly better predictions (`glass`, `scale`, `wine`, and `xd6`). The decision tree predictor is slightly more precise for `car` and `breast`. The largest difference between both classifiers emerges for the `zoo` data set. Interestingly, the graph learners do not perform any joins in this classification problem. The deviations can thus be traced back to the different impurity measures applied when growing the tree. Whereas the decision graph algorithm uses the MML principle to choose the splitting attribute at a specific node, the decision tree classifier employs common entropy measures (in this case: gain ratio) to select the best split.

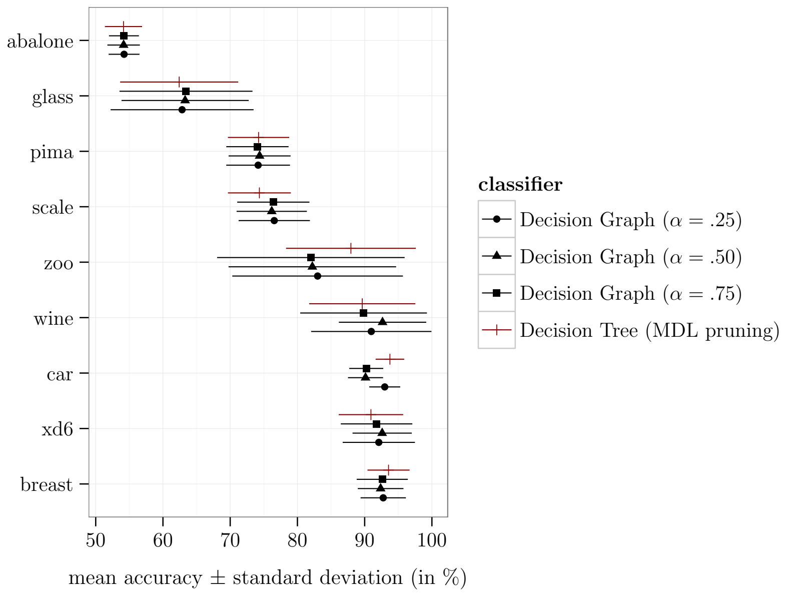


Fig. 8: Results of 10-fold cross-validation repeated 10 times.

5 Conclusion

This paper presented a decision graph algorithm for supervised concept learning. It was argued that decision graphs are superior to decision trees in the case of disjunctive target concepts. The empirical results partly support this suggestion. For the 3rd MONK's problem data set, the decision graph classifier was more robust and gave a better representation of the target function than the decision tree algorithm with MDL pruning. For several other data sets, the graph algorithm performed equally or slightly better than the tree classifier. Only for the `zoo` data set, the graph algorithm was clearly inferior. However, this result was not due to the graph representation of the target function.

Overall, the application of decision graphs seems especially useful when researchers also wish to analyze the substantive relation between the class of instances and their attribute values. As shown above, decision graphs often give a better representation of the underlying concept which might be more interpretable for researchers.

Bibliography

Berthold, Michael R., Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel and Bernd Wiswedel. 2007. KNIME: The Konstanz Information Miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007).* Springer.

Mehta, M, J Rissanen and R Agrawal. 1995. "{MDL}-Based Decision Tree Pruning." *Proc., the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95)* pp. 216–221.

Oliver, J.J., David L. Dowe and C. S. Wallace. 1992. Inferring Decision Graphs Using the Minimum Message Length Principle. In *Proceedings of the 5th Joint Conference on Artificial Intelligence.* pp. 361–367.

Oliver, Jonathan J. 1993. Decision Graphs - An Extension of Decision Trees. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics.* pp. 343–350.

Pagallo, Giulia and David Haussler. 1990. "Boolean Feature Discovery in Empirical Learning." *Machine Learning* 5(1):71–99.

Quinlan, John Ross and Ronald L. Rivest. 1989. "Inferring Decision Trees Using the Minimum Description Length Principle." *Information and Computation* 80(1989):227–248.

Tan, Peter J. 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.* pp. 131–142.

Tan, Peter J. and David L. Dowe. 2003. "MML Inference of Decision Graphs with Multi-Way Joins and Dynamic Attributes." *Proceedings of the 16th Australian Joint Conference on Artificial Intelligence* pp. 269–281.

Wallace, C. S. 2005. *Statistical and Inductive Inference by Minimum Message Length.* Springer-Verlag New York.

Wallace, C. S. and D. L. Dowe. 1999. "Minimum Message Length and Kolmogorov Complexity." *The Computer Journal* 42(4):270–283.

Wallace, C. S. and D. M. Boulton. 1968. "An information measure for classification." *Computer Journal* 11(2):185–194.

Wallace, C. S. and J. D. Patrick. 1993. "Coding Decision Trees." *Machine Learning* 11(1):7–22.