DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9737

MODELLING DECISION TABLES FROM DATA

by

Geert WETS

Jan VANTHIENEN

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9737

MODELLING DECISION TABLES FROM DATA

by

Geert WETS
Jan VANTHIENEN

D/1997/2376/39

MODELLING DECISION TABLES FROM DATA

Geert Wets, Jan Vanthienen

Katholieke Universiteit Leuven, Department of Applied Economic Sciences, Naamsestraat 69, B-3000 Leuven, Belgium, E-mail: {geert.wets, jan.vanthienen}@econ.kuleuven.ac.be

Harry Timmermans

Eindhoven University of Technology, Faculty of Architecture, Building and Planning, Department of Architecture and Urban Planning, P.O. Box 513, Mail station 20, NL-5600 MB Eindhoven, The Netherlands, E-mail: h.j.p.timmermans@bwk.tue.nl

Abstract

On most datasets induction algorithms can generate very accurate classifiers. Sometimes, however, these classifiers are very hard to understand for humans. Therefore, in this paper it is investigated how we can present the extracted knowledge to the user by means of decision tables. Decision tables are very easy to understand. Furthermore, decision tables provide interesting facilities to check the extracted knowledge on consistency and completeness. In this paper, it is demonstrated how a consistent and complete DT can be modelled starting from raw data. The proposed method is empirically validated on several benchmarking datasets. It is shown that the modelled decision tables are sufficiently small. This allows easy consultation of the represented knowledge.

Keywords

Decision tables, verification, visualization

1. Introduction

Currently, there is an urgent need for techniques which can extract knowledge from the data by discovering relations and patterns between the data elements in a (nearly) automatic way. The need for these techniques and tools has created a new field of research called knowledge discovery in data (KDD). KDD can be defined as (Fayyad, Piatetsky-Shapiro & Smyth, 1996):

> "Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."

KDD covers the whole process from raw data until the knowledge, which is extracted from the dataset, will be used. The knowledge extraction process itself is commonly denoted as data mining. While considerable research is devoted to improving data mining algorithms less attention is paid to the verification and validation of the extracted patterns. In this paper, we will show how the output of some classification techniques can be verified using decision tables (DTs). This technique allows some easy checking on consistency and completeness of the extracted knowledge. Furthermore, DTs are quite suited to visualize the extracted knowledge. In this paper, we will show how a computer can learn a proper DT (consistent, complete and sufficiently small) given certain data.

This paper is organized as follows. First, DTs are introduced. Subsequently, it is described how a complete and consistent DT can be modelled starting from raw data. Then, the proposed approach is empirically validated using some benchmarking datasets. Finally, some conclusions are given and some topics for further research are outlined.

2. Decision tables

A DT is a tabular representation used to describe and analyze procedural decision situations, where the state of a number of conditions jointly determines the execution of a set of actions. Not just any representation, however, but one in which all distinct situations are shown as columns in a table, such that every possible case is included in one and only one column (completeness and exclusivity). The tabular representation of the decision situation is characterized by the separation between conditions and actions, on one hand, and between subjects and conditional expressions (states), on the other. Every table column (decision column) indicates which actions should (or should not) be executed for a specific combination of condition states. In this definition, the DT concept is deliberately restricted to the single-hit table, where columns are mutually exclusive. Each possible combination of conditions can be found in one and only one column. Only this type of table allows easy checking for consistency and completeness (Vanthienen and Dries, 1997). Many other variations of the DT concept exist which look similar at first sight. The most important criterion when distinguishing tables is the question whether all columns are mutually exclusive (single-hit versus multiple-hit). In a single-hit table each possible combination of conditions can be found in one and only one column. This facilitates unambiguous use of the table. In multiple-hit tables the same combination of conditions can occur in different columns. As a result, the overview over the columns is lost, and with it, the simplicity of inspection. For these reasons

we do not consider these latter tables to be real DTs. A DT consists of four parts (Codasyl, 1982):

1. The condition subjects are the criteria which are relevant to the decision-making process. They represent the items about which information is needed to take the right decision. Condition subjects are found in the upper left part of the table.
2. The condition states are logical expressions determining the relevant sets of values for a given condition. Every condition has its set of condition states. Condition states are found at the right hand side of the table.
3. The action subjects describe the results of the decision-making process. They are found in the lower left part of the table.
4. The action values are the possible values a given action can take. They are found at the right hand side of the table.

These four parts can be defined more formally:

$CS = \{CS_i\}$ ($i = 1, \ldots, cnum$) is the set of condition subjects;
$CD = \{CD_i\}$ ($i = 1, \ldots, cnum$) is the set of condition domains,

with CD_i the domain of condition i, i.e. the set of all possible values of condition subject CS_i,

$CT = \{CT_i\}$ ($i = 1, \ldots, cnum$) is the set of condition state sets,

with $CT_i = \{S_{ik}\}$ ($k = 1,\ldots, n_i$) an ordered set of n_i condition states S_{ik}. Each condition state S_{ik} is a logical expression concerning the elements of CD_i, that determines a subset of CD_i, such that the set of all these subsets constitutes a partition of CD_i (completeness and exclusivity of the condition states);

$AS = \{AS_j\}$ ($j = 1, \ldots, anum$) is the set of action subjects;
$AV = \{AV_j\}$ ($j = 1, \ldots, anum$) is the set of action value sets,

with AV_j = {true (x), false (-), null (.)} the set of action values, which is, in first instance, null for every action subject, for reasons of consistency checking. A '-' value in the condition part means irrelevant. In the action part it means don't execute. A null value means unknown.

A DT is a function from the Cartesian product of the condition states to the Cartesian product of the action values, by which every condition combination is mapped into one (completeness) and only one (exclusivity) action configuration. If each column only contains simple states (no contractions or irrelevant conditions), the table is called an expanded DT. An example is given in Figure 1.

1. Space (S)	S<20			20<=S<40			S>=40		
2. Costs (C)	C<2	2<=C<4	C>=4	C<2	2<=C<4	C>=4	C<2	2<=C<4	C>=4
1. Premium 1	-	-	x	-	x	x	-	x	x
2. Premium 2	x	x	x	x	-	x	-	-	x
	1	2	3	4	5	6	7	8	9

Figure 1: Example of an expanded DT

If it is necessary, columns in an expanded DT can be contracted. Contraction combines columns or groups of columns that only differ in the state value of one condition and that have equal action configurations into respectively one column. It is important to note that contraction does not change the knowledge contained in the DT. Only the format in which it is presented to the user is changed. Contraction is important in order to enhance the effectiveness of the decision-making or to provide a more compact formulation that can serve as a basis for discussion between the expert and the knowledge engineer. The contracted version of the expanded DT of Figure 1 is depicted in Figure 2. There are only five columns in the contracted DT instead of the nine columns in the expanded DT.

1. Costs (C)	C<2		2<=C<4		C>=4
2. Space (S)	S<20 or 20<=S<40	S>=40	S<20	20<=S<40 or S>=40	-
1. Premium 1	-	-	-	x	x
2. Premium 2	x	-	x	-	x
	1	2	3	4	5

Figure 2: Example of a contracted DT

3. Modelling DTs from data

To model a DT three information elements are necessary: conditions, actions and the decision logic. First, conditions, actions and their respective states have to be retrieved from the dataset. Because we want to extract these information elements automatically from a dataset, the data contained in the dataset should satisfy some constraints. All information about an instance in the dataset should be expressed in terms of a list of values of a number of features (also called attributes). One out of these features is the goal attribute (also called the class or the label) or to put it in DT terminology is the action. The other attributes are the conditions. Furthermore, it is necessary that the features are discrete. If continuous features occur, they have to be discretized by splitting up the domain of the feature in non-overlapping partitions. A plethora of discretization methods has been proposed in the literature. For an overview see Dougherty, Kohavi and Sahami (1995). In our experiments we used the algorithm proposed by Fayyad & Irani (1993). Fayyad and Irani describe an algorithm that uses the entropy function to split the continuous space in two partitions. Recursively, more partitions can be created by the algorithm until some stopping criterion is attained.
Furthermore, it is possible that also some irrelevant features in the dataset occur. Because the number of columns in a DT increases exponentially as the number of conditions increases, it is very important only to select the relevant features in the dataset using some feature selection algorithm. In our experiments, the data were pre-processed using the IDTM algorithm (Kohavi, (1995)).

After that the conditions and the actions are obtained the decision logic has to be derived. In a classical DT modelling method, the decision logic is elicited from an expert (e.g., in the form of rules) and subsequently these rules are used to model the DTs (Vanthienen & Wets, 1994). In this paper, however, the decision logic will be extracted from the dataset using some kind of classification technique and then it will be imported into the DT. Several hypothesis spaces can be used to model the decision logic (e.g. rules and decision trees). In our experiments, we used C4.5 (Quinlan, 1993) to obtain the decision logic. C4.5 is a well-known example of a classification tree algorithm. This type of algorithms tries to fit a tree to

a training sample using recursive partitioning. This means that the training set is split into increasingly homogeneous subsets until the leaf nodes contain only cases from a single class. After that the decision tree is obtained, C4.5 allows to transform the decision tree in rules. These rules can be used to model the DT, as will be explained next.

At this point in the development process two major options can be chosen: constructing a DT from the decision logic which reflects all the information present in the decision logic (thus, also, several types of anomalies such as inconsistency and incompleteness); or constructing a DT which is consistent, complete and correct. When the former option is chosen, the expert himself has to decide how the anomalies which are presented in the DT have to be resolved. The latter option proposes a DT with no anomalies to the expert. To construct such a DT, anomalies which are present in the DT have to be removed using some heuristic. However it is not only necessary that those anomalies are not reflected anymore in the DT, but also the constructed DT should be as correct as possible. Of course it is clear that a DT which is completely correct cannot be constructed for real-life problems because the induced decision logic only partly represents that correct hypothesis.

Both options can easily be combined. For each anomaly in the DT the system can make a suggestion, but it is up to the expert to decide whether he agrees to this suggestion. Next, both options will be explained in more detail.

3.1 Construct a DT in a straightforward way

This option will construct the DT using the extracted knowledge, but the system will offer no help to solve possible anomalies which are present in the decision logic. It will only indicate that there exist anomalies in the DT. The decision to resolve these anomalies will be left to the expert.

First, the empty expanded table has to be drawn using the information elements which had been obtained previously. The name of each feature that occurs in the extracted decision logic will occur in the condition stub of the DT. The possible values of each feature which occur in the decision logic will be reflected in the condition entries part of the DT. Based on this information, the empty expanded DT will be constructed such that it is a single hit tree structured DT. According to the definition of an expanded single hit DT, each combination of condition values must be unique and the condition states must not be irrelevant. The condition entries are filled in such a way that a tree structured DT is obtained. This is a DT that can be evaluated top-down by continuously choosing the relevant condition states until a specific column is reached. In this case, the DT is a straightforward representation of the decision tree with all conditions tested in the same order. The tree structure also implies that the combination of condition values occur from left to right in lexicographical order, in other words that the states of the lowest conditions vary first.

The action stub can be filled using the names of the actions which occur in the decision logic. The condition entries part consists of the Cartesian product of the various condition states. Finally, based on the decision logic the action entries are filled. To fill the action entries based on the extracted rules, the user can either choose to perform this operation interactively or in batch. If the user chooses the first option, each time a rule will be added a new DT will be constructed and a V&V step will be performed. As a result, anomalies which are present in the rules will be immediately reflected in the DT. Thus, the user can

immediately correct anomalies in the DT. If the second option (batch mode) is chosen by the user, all rules will be used to construct the DT. Then, the user can start to correct the possible anomalies in the DT.

3.2 Construct a complete and consistent DT

The second option to construct a DT differs from the first option only in the way that the action entries are filled. Hereby, the system will offer the user some help in order to remove the anomalies present in the decision logic. As was already mentioned, a proper DT needs to be consistent, complete and correct. With respect to this last property, recall that it is not possible to construct a DT which is completely correct for most real-life applications, because the induced hypothesis space is only an estimation of the extremely complex real hypothesis space. Therefore we want to emphasize that the DT, which should be constructed based on the proposed approach, should always be checked by the expert whether it reflects the correct decision for each case. Next, we will investigate how we can construct a consistent and complete DT, which approximates best the, in most cases unknown, correct DT.

In order to construct a consistent and complete DT, we have to ensure that for each possible combination of condition values it should be unambiguously specified which actions should be performed for this combination of condition values. However, if the extracted decision logic contains ambiguity for some combinations of condition values, the following question has to be solved. Which actions should be executed for a combination of condition values, given the fact that the decision logic for this combination of condition values specifies ambivalent actions?

To solve this problem we have to look differently at the condition entries part of an expanded single hit DT. Each column in the condition entries part consists of a unique combination of condition values. Therefore, we will consider such a combination of condition values as an unlabelled instance which has to be classified, by means of the induced rules. In the context of the construction of a DT, classification means filling in the proper action entries for this combination of condition values. When the action entries for a combination of condition entries are filled in, this procedure will be repeated for the remaining combinations of condition values in the DT. It depends on the expert whether he will check each suggestion of the system separately, or that he will check the DT after that the system has filled in all necessary action entries. As a consequence, our initial problem can be reformulated into the following formulation: "how can an unlabelled instance be classified by the induced rules?". For any unlabelled instance during classification three situations may happen:

1. the unlabelled instance is classified unambiguously by one or more rules;
2. the unlabelled instance is classified by some rules into a class and at the same time classified by some other rules in another class;
3. the unlabelled instance is not classified at all.

1. THE UNLABELLED INSTANCE IS CLASSIFIED UNAMBIGUOUSLY

Situation 1 poses no problems, since the unlabelled instance is classified unambiguously by one or more rules. However, this does not mean that this unlabelled instance is classified properly. It only means that, given the induced rule set, this unlabelled instance can be

classified unambiguously. Still, it has to be approved by the expert that the unlabelled instance has been classified properly.

2. THE UNLABELLED INSTANCE IS AMBIGUOUSLY CLASSIFIED

In situation 2, more than one rule matches the unlabelled instance and the matching rules specify contradictory actions to be executed. The question is: how should such an unlabelled instance be classified? In the machine learning literature, several approaches are proposed to deal with this problem.

A first approach to the above mentioned problem uses decision lists (Rivest, 1987). A decision list is an ordered list of rules. The earliest rule that matches an unlabelled instance will classify the unlabelled instance. The last rule is a default rule. This rule will classify the unlabelled instance as no other rule does classify the unlabelled instance. Decision lists are the most simple technique to solve the problem of inconsistency. Well-known machine learning algorithms which use decision lists are C4.5 (Quinlan, 1993) and the original CN2 algorithm (Clark & Niblett, 1989).

A second approach to classify an unlabelled instance is used in AQ15 (Michalski, Mozetic, Hong & Lavrac, 1986). AQ15 will select the rules which completely match the unlabelled instance (using AQ15 terminology, "strict matching"). Using an heuristic which uses information about the matching rules, the unlabelled instance will be classified. Therefore, in AQ15 with every rule R an estimate of probability EP(R) is associated. This estimate is defined as follows:

$$EP(R) = \frac{\text{number of examples classified properly by rule } R}{\text{total number of training examples}}$$

Based on these EP(R_i), the number EP(C), describing the class C, is computed as the probabilistic sum of all EP(R_i) matching C. Note that by doing so the EP(R_i) are treated as if they were probabilities, since the probabilistic sum computes the probability of the disjunction of n events (here n rules). But one has to keep in mind that in fact the EP(R_i) are only estimates of probabilities. The unlabelled instance is classified as belonging to the class which has the highest probabilistic sum.

Another approach to classify an unlabelled instance is presented in Holland, Holyoak & Nisbett (1986) under the name "bucket brigade algorithm". The same strategy has been adopted by the LERS system (Grzymala-Busse, 1994). In this approach, the decision to which class an unlabelled instance belongs is made on the basis of three factors: strength, specificity and support. The meaning of these factors is as follows:

- The strength factor measures how well a rule has performed in the past (e.g., on a training set).
- Specificity measures the relevance of a rule. The more detailed the rule's condition part, the greater its specificity. Specificity of a rule is equal to the number of attributes in the condition part of a rule.
- Support is defined as the sum of scores of all matching rules from the class. The score for a rule is calculated by multiplying the strength factor for the rule with the specificity of the rule. An unlabelled instance will be classified as belonging to the class with the highest support.

In our experiments the DTs were modelled using decision lists. Currently, we are investing also the other techniques to model DTs from data.

3. THE UNLABELLED INSTANCE IS NOT CLASSIFIED AT ALL

In this situation, there is no rule which exactly matches the unlabelled instance. This means that not all attribute values of the rule are matched by their counterparts in the unlabelled instance. As a consequence, the DT that will be modelled will not be complete. Because it is an important goal to construct a complete DT, this situation is unsatisfactory.

A first solution to this problem uses a default rule. A default rule is a rule which will classify the unlabelled instance if all other rules fail to do so. It can be seen as a last resort. A well-known induction algorithm which uses a default rule is C4.5.

A second solution is used in AQ15. AQ15 will classify an unlabelled instance using partial matching (using AQ15 terminology, "analogical matching"). To this end an heuristic is proposed which uses information about the partial matching rules. Based on this heuristic, the unlabelled instance will be classified. First, a measure of fit for each attribute value (a_i) occurring in a rule and each attribute value (f_i) occurring in an unlabelled instance. This measure of fit MF(a_i, f_i) can be defined as follows. First, a measure of fit MF(R) for every rule is calculated. This measure of fit is defined as follows:

if $a_i = f_i$ then MF(a_i, f_i) = 1
else MF(a_i, f_i) = 1/|number of the attribute's possible values|

Note that in AQ15 it is possible that the left hand side of a rule contains expressions of the form $(a, f_1 \vee f_2 \vee \ldots \vee f_k)$. This expression indicates that the value for the attribute a may take k different values. As a result, MF(a_i, f_i) will become equal to k / |number of the attribute's possible values|.

If there are n attributes in a rule the measure of fit for a rule can be defined as:

$$\mathrm{MF}(R) = (\prod_{i=1}^{n} MF(a_i, f_i)) * (\frac{\text{strength of a rule}}{\text{total number of training examples}})$$

In this expression, $\prod_{i=1}^{n} MF(a_i, f_i)$ is a weighting factor. This factor indicates how good the unlabelled instance matches the rule. Based on the measure of fit for the rules, a measure of fit for a class C can easily be computed. The same procedure as that was used to compute EP(C) is taken. Given n partial matching rules, the measure of fit for a class is the probabilistic sum of all MF(R).

A third solution is presented in LERS. To compute the support in case of partial matching besides strength and specificity an additional factor is taking into account, the matching factor. This factor is defined as follows:

$$Matching(R) = \frac{\text{number of matched attribute values in rule } R}{\text{total number of attributes in rule } R}$$

Thus, the support for a rule R can now be calculated as follows:

$$Support(R) = Matching(R) * Strength(R) * Specificity(R)$$

Subsequently, using the support which was calculated for each rule *R*, the support for a class *C* can be computed. The support for each class *C* is calculated by taking the summation of the support of rules with respect to the class *C*.

In our experiments we used a default rule to avoid completeness. Currently, we are also experimenting with the other outlined techniques.

4. Empirical evaluation

To illustrate the proposed approach, it was tested on seven datasets. All the datasets used in this section came from the UC Irvine repository (Merz & Murphy, 1996). Prior to the analysis, instances with missing values were removed from the training set. In the next table, an overview of the selected datasets is given.

Dataset	Features	Classes	Training size	Test size
Breast	10	2	699	10-fold stratified CV
Cleve	13	2	303	10-fold stratified CV
Aucrx	15	2	690	10-fold stratified CV
Pima	8	2	768	10-fold stratified CV
Sick	25	2	3163	10-fold stratified CV
Monk1	6	2	124	432
Monk3	6	2	122	432

Table 1: Summary of datasets used

First, the datasets were pre-processed (discretization using Fayyad's and Irani's method and feature selection using IDTM). Then, the decision logic was induced using C4.5. Finally, the DTs were constructed and contracted. In the experiment, we used 10-fold stratified cross-validation. In general, this method allows that we accurately measure the estimated accuracy. Because 10-fold stratified cross-validation was used ten rule sets for each dataset exist. If not all folds classify an example in the same way, the class which occurs most frequently is used. In case of a tie, the default class is used. In Figure 3 the modelled DT is depicted for the dataset Breast.

1. uniformcellsha (U)	1<=U<3			3<=U<5			5<=U
2. barenuclei (B)	1<=B<3 or 3<=B<6	6<=B		1<=B<3		3<=B<6 or 6<=B	-
3. mitoses (M)	-	1<=M<2	2<=M	1<=M<2	2<=M	-	-
1. benign	x	-	x	-	x	-	-
2. malignant	-	x	-	x	-	x	x
	1	2	3	4	5	6	7

Figure 3: DT for Breast dataset

It can be seen that this DT is sufficiently small, so that it can be interpreted easily. Also the other DTs were quite small as is depicted in the next table. The major reasons why the DTs are so small is pre-processing by discretization and feature selection and optimization of the DT by table contraction. It can easily be seen that the impact of this reduction is enormous with respect to the number of possible columns in the DT. For example, for the dataset breast the number of possible columns in the DT before discretization and feature selection is a staggering 10^9. The expanded DT after discretization and feature selection shows only

eighteen columns. Table contraction reduces the number of columns even further. The contracted table shows only seven columns.

Dataset	# columns (expanded)	# columns (contracted)
Breast	18	7
Cleve	24	8
Aucrx	72	2
Pima	32	9
Sick	6	3
Monk1	36	8
Monk3	12	3

Table 2: Number of columns in the modelled DTs

One may argue that the number of columns is largely reduced. But, this might be at the cost of a significant reduction in expected accuracy. Therefore, in the next table, results of C4.5rules on the raw data are compared with the results of the classification accuracy attained by the DTs.

Dataset	Accuracy of C4.5rules	Accuracy of the DT
Breast	95.76	96.47
Cleve	75.34	82.67
Aucrx	83.93	85.20
Pima	72.78	77.92
Sick	97.23	96.93
Monk1	91.67	100.00
Monk3	96.30	97.22
AVG	86.12	89.87

Table 3: Comparison of accuracy

In this table, it can be seen that for most datasets the accuracy actually improves after feature selection and discretization. These results show that feature selection and discretization improve greatly the comprehensibility of the DTs and moreover, the accuracy of the knowledge contained in the DTs gives still a very good estimation of the actual unknown distribution of the data.

5. Conclusion and future research

Originally, DTs were constructed based on some knowledge provided by an expert or some piece of regulation. In this paper, we have demonstrated that it is possible to model a complete and consistent DT from data. Therefore, a DT was interpreted as a set of examples which have to be appropriately classified using some knowledge induced from the dataset. Our proposed approach was empirically validated and it was shown that the modelled DTs are small enough in order to facilitate consultation.

In this paper several techniques to model a complete and consistent DT were presented. However, in our experiments only one technique was used so far. Although the results, as we have demonstrated, are satisfactory it would be interesting to compare all the proposed techniques to model DTs from data. Currently, such experiments are carried out.

6. References

Clark, P. & Niblett, T. (1989), The CN2 induction algorithm, *Machine Learning 3*, pp. 261-283.

Codasyl (1982), *A Modern Appraisal of Decision Tables*, Report of the decision table task group, ACM, New York (N. Y.).

Dougherty, J., Kohavi, R. & Sahami, M. (1995), Supervised and unsupervised discretization of continuous features, *Machine Learning: Proc. of the 12th Intl. Conf.*, pp. 194-202.

Fayyad, U. M. & Irani, K. B. (1993), Multi-interval discretization of continuous-valued attributes for classification learning, *Proc. of the 13th Intl. Joint Conf. on Artificial Intelligence*, pp. 1022-1027.

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996), From data mining to knowledge discovery: An overview, in: *Advances in Knowledge Discovery and Data Mining*, Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. (eds.), MIT Press, Menlo Park (CA), pp. 1-34.

Holland, J. H., Holyoak, K. J. & Nisbett, R. E. (1986), *Induction: Processes of Inference, Learning and Discovery*, MIT Press, Cambridge (MA).

Grzymala-Busse, J. W. (1994), Managing uncertainty in machine learning from examples, *Proc. of the Workshop on Intelligent Information Systems III*, pp. 70-84.

Kohavi, R. (1995), The power of decision tables, *Proc. of the European Conf. on Machine Learning*, Lecture Notes in Artificial Intelligence 914, Springer Verlag, Berlin, pp. 174-189.

Merz, C. J. & Murphy, P. M. (1996), *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html], Irvine, CA: University of California, Dept. of Information and Computer Science.

Michalski, R. S., Mozetic, I., Hong, J. & Lavrac, N. (1986), The multi-purpose incremental learning system AQ15 and its testing application to three medical domains, *Proc. of the 5th National Conf. on AI*, pp. 1041-1045.

Quinlan, J. R. (1993), *C4.5 Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo (CA).

Rivest, R. L. (1987), Learning decision lists, *Machine Learning 2*, pp. 229-246.

Vanthienen, J. & Dries, E. (1997), Decision tables: Refining concepts and a proposed standard, *Comm. of the ACM,* to appear.

Vanthienen, J. & Wets, G. (1994), From decision tables to expert system shells, *Data & Knowledge Engineering 13*, pp. 265-282.