A Decision Tree Plug-In for DataEngine™

Christian Borgelt

Dept. of Knowledge Processing and Language Engineering
Otto-von-Guericke-University of Magdeburg
Universitätsplatz 2, D-39106 Magdeburg, Germany

e-mail: borgelt@iws.cs.uni-magdeburg.de

Abstract: Inducing decision trees with a top-down algorithm is a well-known and widely used method to construct classifiers from a set of sample cases. In this paper we review this technique in order to demonstrate its relative simplicity and its power to produce comprehensible results. Since the task to induce a classifier from data turns up frequently in applications (e.g. credit assessment, disease detection etc.) no commercial data analysis tool can do without offering a decision tree induction module. Nevertheless, the well-known data analysis program DataEngine™ has until recently suffered from lacking such a module. This drawback is now removed by a plug-in consisting of a set of user-defined function blocks we implemented and which we describe in this paper.

1 Introduction

Decision trees are a well-known type of classifiers. Classifiers, in turn, are programs which automatically classify a case or an object, i.e. assign it according to its features to one of several given classes. For example, if the cases are patients in a hospital, the attributes are properties of the patients (e.g. sex, age etc.) and their symptoms (e.g. fever, high blood pressure etc.), the classes may be diseases or drugs to administer.

Decision trees are classifiers which — as the name already indicates — have a tree-like structure. To each leaf a class, to each inner node an attribute (or feature) is assigned. There can be several leaves associated with the same class and several inner nodes associated with the same attribute. The descendants of the inner nodes are reached via edges to each of which a value of the attribute associated with the node is assigned. Each leaf represents a decision "The case considered belongs to class c.", where c is the class associated with the leaf. Each inner node corresponds to an instruction "Test attribute A and follow the edge to which the value observed is assigned!", where A is the attribute associated with the node. A case is classified by starting at the root of the tree and executing the instructions in the inner nodes until a leaf is reached, which then states a class.

From the above description it is obvious that decision trees are very simple to use. Unfortunately, it is not quite as simple to construct them manually. Especially if the number of possible test attributes is large and the available knowledge about the underlying relations between the classes and the test attributes is vague, manual construction can be tedious and time consuming. However, if a database of sample cases is available, one can try an automatic induction [5, 18, 19]. The usual approach is a top-down process (TDIDT — top-down induction of decision trees), which uses a "divide and conquer" principle together with a greedy selection of test attributes according to the value ascribed to them by an evaluation measure. In section 2 we illustrate this approach using a simple (artificial) medical example.

Since the success of the induction algorithm depends considerably on the attribute selection measure used, in section 3 we list a large variety of such measures. However, limits of space prevent us from discussing in detail the ideas underlying them. Which of these measures yields the best results cannot be stated in general, but depends on the application. Therefore all of these measures can be selected for the decision tree induction function block of the DataEngine™ plug-in we describe in section 4. In addition to the induction function block this plug-in consists of functions blocks for pruning a decision tree, for executing a decision tree to classify a set of cases and for computing a confusion matrix (which is useful to assess the quality of a learned classifier).

2 Induction of Decision Trees

As already remarked above the induction of decision trees from data rests on a "divide and conquer" principle together with a greedy selection of the attributes to test: From a given set of classified case descriptions the conditional frequency distributions of the classes given the attributes used in the case descriptions are computed. These distributions are evaluated using some measure and the attribute yielding the best value is selected as the next test attribute. This is the greedy

No	Sex	Age	Blood Pressure	Drug
1	male	20	normal	A
2	female	73	normal	B
3	female	37	high	A
4	male	33	low	B
5	female	48	high	A
6	male	29	normal	A
7	female	52	normal	B
8	male	42	low	B
9	male	61	normal	B
10	female	30	normal	A
11	female	26	low	B
12	male	54	high	A

Table 1: This table contains patient data together with an effective drug (effective w.r.t. some unspecified disease). To find a decision tree for the effective drug, the conditional distributions of the drugs given the patients' features are examined (see table 2).

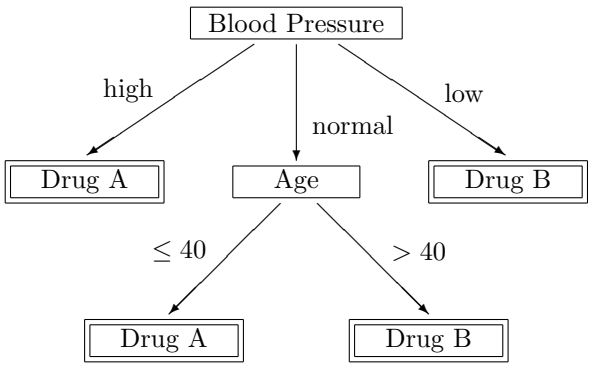

Figure 1: The learned decision tree. First the blood pressure of the patient is determined. If it is low or high, the effective drug can be stated directly. If it is normal, we inquire for the age of the patient and thus can derive the effective drug in this case, too.

part of the algorithm. Then the case descriptions are divided according to the values of the chosen test attribute and the procedure is applied recursively to the resulting subsets. This is the "divide and conquer" part of the algorithm. The recursions stops, if either all cases of a subset belong to the same class, or no attribute yields an improvement of the classification, or there are no attributes left for a test. We illustrate this procedure with a simple example and state the induction algorithm in pseudo code.

2.1 A Simple Example

Table 1 shows the features of twelve patients — sex, age, and a qualitative statement of the blood pressure — together with a drug, which for the patient has been effective in the treatment of some unspecified disease. Neglecting the features of the patients the effective drug can be predicted only with a rate of success of 50%, since drug A as well as drug B were effective in six cases. Because such a situation is unfavourable for future treatments, we try to learn a decision tree, which will (hopefully) allow us to derive the effective drug from the features of a patient.

To this end we consider all conditional distributions of the effective drugs given the available features (see table 2). It is obvious that the patient's sex is without any influence, since for male as well as for female patients both drugs were effective in half of the cases (thus being politically correct, i.e. here: non-sexist). The patients age yields a better result: below forty years of age drug A has been effective in four out of six cases. Over forty years of age the same holds for drug B. Hence, the success rate is 67%. However, testing the blood pressure yields an even better result: If it is high, drug A, if it is low, drug B is the correct drug. Only if the blood pressure is normal, the prediction is not improved. The overall success rate is 75%.

Since the blood pressure allows us to determine the effective drug with the highest rate of success, it is chosen as the first test attribute and placed at the root of the decision tree. The case descriptions of the table are divided according to the values they contain for this attribute. Since the effective drug is definite for patients with low or high blood pressure, these cases need not be considered further. For the patients with normal blood pressure we test again the conditional distribution of the effective drug given the patient's age. Dividing the patients in those younger than forty and those older, separates the cases in which drug A was effective from those in which drug B was effective. Thus a reliable method to determine the effective drug is found. The corresponding decision tree, which can be read directly from table 2 (far right), is shown in figure 1.

2.2 The Induction Algorithm

The general algorithm to induce a decision tree from data is shown in figure 2 in a pseudo code similar to Pascal. In the first part of the algorithm for each attribute the frequency distribution of its values and the classes is determined. From this distribution the value of an evaluation measure is computed. The attribute with the highest value is stored in the variable *best_A*. This is a crucial step in the algorithm, since a wrong assessment of the attributes and thus a bad choice for the test attribute can severely diminish the classifier's performance. (More about evaluation measures can be found in section 3.) In the second part of the algorithm either a leaf or a test node is created — depending on the outcome of the first part. If a test node is cre-

No	Sex	Drug
1	male	A
6	male	A
12	male	A
4	male	B
8	male	B
9	male	B
3	female	A
5	female	A
10	female	A
2	female	B
7	female	B
11	female	B

No	Age	Drug
1	20	A
11	26	B
6	29	A
10	30	A
4	33	B
3	37	A
8	42	B
5	48	A
7	52	B
12	54	A
9	61	B
2	73	B

No	Blood Pr.	Drug
3	high	A
5	high	A
12	high	A
1	normal	A
6	normal	A
10	normal	A
2	normal	B
7	normal	B
9	normal	B
4	low	B
8	low	B
11	low	B

No	Blood Pr.	Age	Drug
3	high	37	A
5	high	48	A
12	high	54	A
1	normal	20	A
6	normal	29	A
10	normal	30	A
7	normal	52	B
9	normal	61	B
2	normal	73	B
11	low	26	B
4	low	33	B
8	low	42	B

Table 2: The conditional distributions of the effective drug given the sex (far left), the age (left, divided into "less than 40" and "over 40") and the blood pressure (right) of the patients. Obviously the blood pressure yields the best result. If a test of the age is added in those cases in which the blood pressure is normal (far right), the prediction of the effective drug becomes perfect. The corresponding decision tree, which can be read directly from the table, is shown in figure 1.

```
function grow_tree (S : set of cases) : node;
begin
    best_v := WORTHLESS;
    for all untested attributes A do
        compute frequencies N_ij, N_i., N_.j
            for 1 ≤ i ≤ n_C and 1 ≤ j ≤ n_A;
        compute value v of a selection measure
            using N_ij, N_i., N_.j;
        if v > best_v
        then best_v := v;
            best_A := A;
        end;
    end
    if best_v = WORTHLESS
    then create leaf node n;
        assign majority class of S to n;
    else create test node n;
        assign test on attribute best_A to n;
        for all a ∈ dom(best_A) do
            n.child[a] := grow_tree(S|_{best_A=a});
        end;
    end;
    return n;
end;    (* grow_tree() *)
```

Figure 2: The TDIDT (top-down induction of decision trees) algorithm.

ated, the case descriptions are divided according to their value for the chosen test attribute and for each resulting subset the function grow_tree is called recursively.

To simplify the algorithm we assumed in this description that all attributes have a finite number of symbolic values. Integer or real-valued attributes can be processed by sorting the occurring values and choosing a cut value for each pair of consecutive values (e.g. the arithmetic mean of the two values). Using this cut value an (artificial) symbolic attribute with values "greater than cut value" and "less than cut value" is created. The best cut value, i.e. the one whose corresponding symbolic attribute is rated best by the chosen evaluation measure, is selected to represent the numeric attribute.

During the recursive descent already tested symbolic attributes are marked, since another test of these attributes is obviously pointless: Dividing the cases leads to all cases having the same value for a tested attribute in the deeper levels of the recursion. Integer and real-valued attributes, however, are not marked, since deeper down in the recursion a different cut value may be chosen and thus the range of values may be subdivided further.

3 Attribute Selection Measures

As already indicated in the introduction and substantiated by the description of the general induction algorithm in the preceding section, the success of the induction of a decision tree from data depends to a high degree on the attribute selection measure used. Several years of research not only in decision tree induction but also in the closely related area of inducing Bayesian networks from data has lead to a large variety of evaluation measures, which draw from a substantial set of ideas to assess the quality of an attribute. Unfortunately, limits of space prevent us from discussing in detail these measures and the ideas underlying them. Hence we only give a list:

- information gain I_{gain} [16, 7, 18]
  (mutual information/cross entropy)
- information gain ratio I_{gr} [18, 19]
- symmetric information gain ratio I_{sgr} [17]
- Gini index [5, 21]
- symmetric Gini index [22]
- modified Gini index [13]
- relief measure [12, 13]
- χ^2 measure
- weight of evidence [14]
- relevance [1]
- K2 metric [8, 11]
- BDeu metric [6, 11]
- minimum description length with coding based on relative frequencies l_{rel} [14]
- minimum description length with coding based on absolute frequencies l_{abs} [14]
  (closely related to the K2 metric)
- stochastic complexity [15, 20]
- specificity gain S_{gain} [10, 3]
- (symmetric) specificity gain ratio S_{gr} [3]

It may be worth noting that the K2 metric and the BDeu metric were originally developed for learning Bayesian networks and that the specificity measures are based not on probability or information theory but on possibility theory — an alternative theory for reasoning with imperfect knowledge that is closely connected to fuzzy set theory. A reader who is interested in more detailed information about the measures listed above may consult [3] or [4] (the latter is available in German only).

Unfortunately, no general rule can be given which measure should be chosen. Although some measures (e.g. the information gain ratio and the minimum description length measures) perform slightly better on average, all have their strengths and weaknesses. For each measure there are specific situations in which it performs best and hence it can pay to try several measures.

4 The DataEngine™ Plug-In

We implemented a powerful decision tree induction algorithm as a plug-in for the well-known data analysis program DataEngine™ in order to improve this esteemed tool even further. This plug-in consists of four user-defined function blocks:

grow — Grow a decision tree.
This function block receives as input a table of classified sample cases and grows a decision tree. The data types of the table columns (either symbolic or numeric) can be stated in the unit fields of the table columns, which can also be used to instruct the algorithm to ignore certain columns. Although tables passed to user-defined functions blocks may not contain unknown values, this function block provides a facility to specify which table fields should to considered as unknown: In the configuration dialog you may enter a value for the lowest known value. All values below this value are considered to be unknown. In addition the configuration dialog lets you choose the attribute selection measure (see the preceding section for a list), whether the measure should be weighted with the fraction of known values (to take into account the lesser utility of rarely known attributes), whether the algorithm should try to form subsets on symbolic attributes, a maximal height for the decision tree to be learned, and the name of a file into which the learned decision tree should be saved.

prune — Prune a learned decision tree.
This function block receives as input a learned decision tree stored in a file and a table of classified sample cases, which may or may not be the table the decision tree was learned from. It prunes the decision tree using the table applying one of two pruning methods (either pessimistic pruning or confidence level pruning), which are governed by a parameter that can be entered in the configuration dialog. In addition, the configuration dialog lets you enter a maximal height for the pruned tree and (to be able to deal with unknown values, see above), a lowest known value. The pruned decision tree is written to another file, whose name can also be specified in the configuration dialog.

exec — Execute a learned decision tree.
This function block receives as input a learned (and maybe pruned) decision tree stored in a file and a table of cases. It executes the decision tree for each case in the table and adds to it a new column containing the class predicted by the decision tree. The configuration dialog lets you enter the name of the classification column and (just as described for the two blocks above) a lowest known value.

xmat — Compute a confusion matrix.
This function blocks receives as input a table. Its configuration dialog lets you enter the names of two columns for which a confusion matrix shall be determined. It generates a table containing the confusion matrix (either with absolute or relative numbers) and the sums over lines and columns (excluding the diagonal elements).

All function blocks that deal directly with decision trees, i.e. the blocks *grow*, *prune*, and *exec* also comprise a decision tree viewer which lets you navigate through a learned decision tree using the well-known MS Windows tree view control (used, for example, in the MS Windows explorer to visualize the hierarchic

file system). Hence you need not accept the learned classifier as a black box (as is usually the case for e.g. neural networks), but you can inspect how an induced decision tree arrives at its results.

5 Summary

In this paper we reviewed the well-known TDIDT algorithm for inducing a decision tree from a set of sample cases in order to demonstrate its relative simplicity and its power to produce comprehensible results, which makes it a useful tool for a lot of data analysis task. We presented an implementation of a decision tree induction algorithm as a plug-in for the well-known data-analysis tool DataEngine™. This implementation can make use of a large variety of evaluation measures and, in a separate function block, provides a flexible pruning facility. Two more function blocks allow you to execute a learned decision tree to classify a set of cases and to compute a confusion matrix to assess the quality of the learned classifier.

References

[1] P.W. Baim. A Method for Attribute Selection in Inductive Learning Systems. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-10:888-896, 1988

[2] C. Borgelt, J. Gebhardt and R. Kruse. Concepts for Probabilistic and Possibilistic Induction of Decision Trees on Real World Data. *Proc. of the EUFIT'96*, Vol. 3:1556–1560, 1996

[3] C. Borgelt and R. Kruse. Evaluation Measures for Learning Probabilistic and Possibilistic Networks. *Proc. of the FUZZ-IEEE'97*, Vol. 2:pp.669–676, Barcelona, Spain, 1997

[4] C. Borgelt and R. Kruse. Attributauswahlmaße für die Induktion von Entscheidungsbäumen: Ein "Uberblick. (written in German) In: G. Nakhaeizadeh, ed. Data Mining: Theoretische Aspekte und Anwendungen. pp. 77-98 Physica-Verlag, Heidelberg, Germany 1998

[5] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*, Wadsworth International, Belmont, CA, 1984

[6] W. Buntine. Theory Refinement on Bayesian Networks. *Proc. 7th Conf. on Uncertainty in Artificial Intelligence*, pp. 52–60, Morgan Kaufman, Los Angeles, CA, 1991

[7] C.K. Chow and C.N. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Trans. on Information Theory* 14(3):462–467, IEEE 1968

[8] G.F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning* 9:309–347, Kluwer 1992

[9] J. Gebhardt and R. Kruse. Learning Possibilistic Networks from Data. *Proc. 5th Int. Workshop on AI and Statistics*, 233–244, Fort Lauderdale, 1995

[10] J. Gebhardt and R. Kruse. Tightest Hypertree Decompositions of Multivariate Possibility Distributions. *Proc. Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems*, 1996

[11] D. Heckerman, D. Geiger, and D.M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning* 20:197–243, Kluwer 1995

[12] K. Kira and L. Rendell. A Practical Approach to Feature Selection. *Proc. 9th Int. Conf. on Machine Learning (ICML'92)*, pp. 250–256, Morgan Kaufman, San Franscisco, CA, 1992

[13] I. Kononenko. Estimating Attributes: Analysis and Extensions of RELIEF. *Proc. 7th Europ. Conf. on Machine Learning (ECML'94)*, Springer, New York, NY, 1994

[14] I. Kononenko. On Biases in Estimating Multi-Valued Attributes. *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining*, 1034–1040, Montreal, 1995

[15] R.E. Krichevsky and V.K. Trofimov. The Performance of Universal Coding. *IEEE Trans. on Information Theory*, 27(2):199–207, 1983

[16] S. Kullback and R.A. Leibler. On Information and Sufficiency. *Ann. Math. Statistics* 22:79–86, 1951

[17] R. Lopez de Mantaras. A Distance-based Attribute Selection Measure for Decision Tree Induction. *Machine Learning* 6:81–92, Kluwer 1991

[18] J.R. Quinlan. Induction of Decision Trees. *Machine Learning* 1:81–106, 1986

[19] J.R. Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1993

[20] J. Rissanen. Stochastic Complexity. *Journal of the Royal Statistical Society (Series B)*, 49:223-239, 1987

[21] L. Wehenkel. On Uncertainty Measures Used for Decision Tree Induction. *Proc. IPMU*, 1996

[22] X. Zhou and T.S. Dillon. A statistical-heuristic Feature Selection Criterion for Decision Tree Induction. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-13:834–841, 1991