Advanced Fuzzy Clustering and Decision Tree Plug-Ins for DataEngine™

Christian Borgelt and Heiko Timm

Otto-von-Guericke University of Magdeburg
Faculty of Computer Science (FIN-IWS)
Universitätsplatz 2, D-39106 Magdeburg, Germany
{christian.borgelt,heiko.timm}@cs.uni-magdeburg.de

Abstract. Although a large variety of data analysis tools are available on the market today, none of them is perfect; they all have their strengths and weaknesses. In such a situation it is important that a user can enhance the capabilities of a data analysis tool by his or her own favourite methods in order to compensate for shortcomings of the shipped version. However, only few commercial products offer such a possibility. A rare exception is DataEngine™, which is provided with a well-documented interface for user-defined function blocks (plug-ins). In this paper we describe three plug-ins we implemented for this well-known tool: An *advanced fuzzy clustering plug-in* that extends the fuzzy c-means algorithm (which is a built-in feature of DataEngine™) by other, more flexible algorithms, a *decision tree classifier plug-in* that overcomes the serious drawback that DataEngine™ lacks a native module for this highly important technique, and finally a *naive Bayes classifier plug-in* that makes available an old and time-tested statistical classification method.

1 Introduction

The rapidly growing amount of data that is collected and stored nowadays has created a need for intelligent and easy to use data analysis software, since we are already far beyond the point up to which a "manual" analysis is possible. Several companies have responded to this need and as a consequence a variety of tools and software packages, each with its own strengths and weaknesses, is available today. In this paper we concentrate on one of them, namely DataEngine™[1], a data analysis tool that is strongly oriented towards soft computing methods [1]. Of course, it also offers basic statistical techniques, but its main strengths are fuzzy logic based methods (fuzzy rule bases, fuzzy c-means clustering) and artificial neural networks (multi-layer perceptrons, (fuzzy) Kohonen feature maps). For an impression of this program, see Fig. 1, which shows its main window.

An important strength of DataEngine™ is that it is equipped with what may be called a "graphical programming interface", although it is not the only

[1] DataEngine™ is distributed by Management Intelligenter Technologien GmbH, Aachen, Germany. It is available for MS Windows 95/98™ and MS Windows NT™.

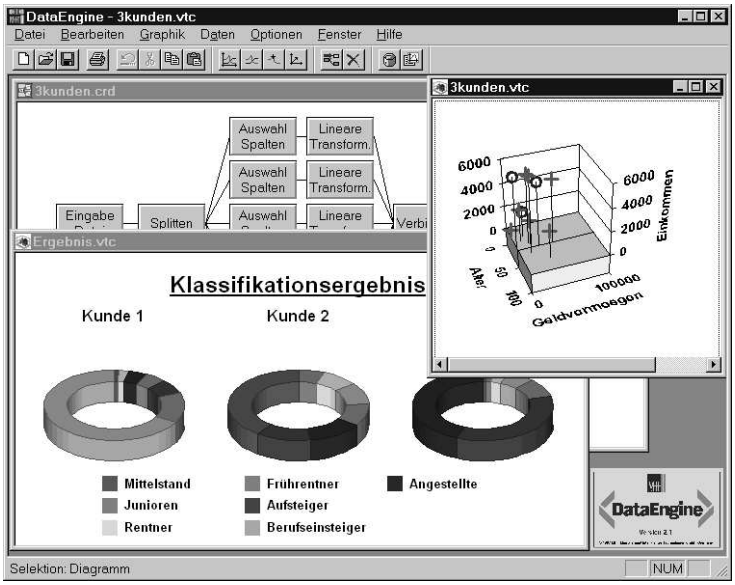

Fig. 1. DataEngine™ main window with some example visualizations. In the background there is a so-called "card" specifying a data flow.

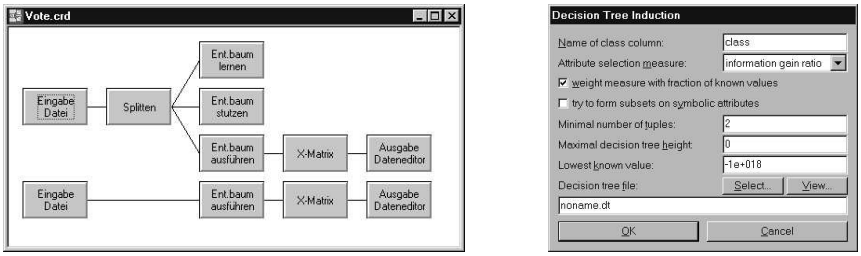

Fig. 2. A so-called "card" of DataEngine™ (left), i.e. a workspace in which a data flow can be specified with a graphical macro language consisting of function blocks, and the configuration dialog of a function block (right).

tool on the market that uses such an interface. This interface allows a user to specify a data flow in a kind of graphical macro language that consists of function blocks, each of which—depending on its type—aggregates, modifies, analyses, or visualizes the data or constructs a model. The basic version of DataEngine™ offers a rich variety of techniques in the form of function blocks, including data access (flat files or ODBC), data preprocessing, descriptive statistics, 2D and 3D visualizations, training and executing neural networks etc. Examples of the visualization capabilities of Data Engine™are shown in Fig. 1.

An example of a so-called "card", i.e. a workspace in which a data flow can be specified graphically, is shown on the left in Fig. 2. It describes the data flow for the induction, pruning, and testing of a decision tree. On the left there are two function blocks labelled "Eingabe Datei"[2] which provide access to data files, the upper to the training, the lower to the test data. The function block to the right of the upper input block (labelled "Splitten") splits the data flow, so that the same input data can be fed into several function blocks. The two topmost function blocks in the middle column induce ("Ent.baum lernen") and prune ("Ent.baum stutzen") a decision tree on the data, the other two ("Ent.baum ausführen") execute the pruned decision tree on the training and the test data.[3] (That it is the pruned decision tree that is executed cannot be read directly from this card, though, because it is passed via a file. DataEngine™ does not offer an explicit facility to pass models between function blocks.) The result of each execution is analysed by computing confusion matrices ("X-Matrix"), which are then displayed in a data editor ("Ausgabe Dateneditor"). Each function block is equipped with a so-called configuration dialogue, in which parameters of the block can be entered. As an example the configuration dialogue of the decision tree induction function block is shown on the right in Fig. 2.

Despite the rich variety of methods it offers, DataEngine™, like any other data analysis tool, is far from perfect. Fortunately—and this is a rare exception—a user need not be content with the capabilities of the shipped version, but can compensate detected shortcomings by enhancing the program with so-called *user-defined function blocks* or *plug-ins*. DataEngine™ supports a well-documented interface for Microsoft Visual C/C++™, Borland C/C++™ and Borland Delphi™ (other languages are also possible, provided a MS Windows™ dynamic link library can be created) via which the data tables of DataEngine™ can be accessed and processed with any user-defined method. We have made extensive use of this interface: up to now we have implemented three plug-ins to overcome weaknesses of the original product or to extend its capabilities.

The first plug-in is a fuzzy clustering module [2], where *clustering* is the process of finding groups of similar cases or objects in a given dataset. The term "fuzzy" indicates that the grouping is not crisp, i.e. the cases or objects are not assigned to one (and only one) cluster each, but may belong (with different degrees) to more than one cluster. In many applications such a "softening" of the boundaries between clusters leads to better results.

Although DataEngine™ is equipped with a built-in fuzzy clustering module, it is rather limited in this respect, since it only offers the standard fuzzy c-means algorithm. However, there are several other fuzzy clustering algorithms that are much more flexible with respect to the shape and the size of the clusters. In order to make these algorithms more widely available we implemented these methods as a plug-in. This plug-in can be obtained from the MIT GmbH, which distributes it under the name "Advanced Fuzzy Clustering".

[2] Unfortunately we only have a German version of DataEngine™, so all function blocks are labelled in German.

[3] These decision tree functions blocks are part of the plug-in described in section 3.

The other two plug-ins are classifiers. A *classifier* is a program which automatically classifies a case or an object, i.e. assigns 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. The automatic induction of a classifier from a dataset of sample cases is a very frequent task in applications.

The best-known type of classifier is, of course, the decision tree. However, although a decision tree induction module cannot be dispensed with in data analysis—several data analysis tools even rely exclusively on this comprehensible and often highly successful method—DataEngine™ lacks a native module for this type of classifier. In order to overcome this serious drawback, we implemented the well-known top-down induction method for decision trees [3]. The "DecisionXpert" plug-in, which is offered by the MIT GmbH as an add-on for DataEngine™, is based on this implementation.

The second classifier plug-in uses the old and time-tested naive Bayes approach of classical statistics to construct a classifier and to classify new cases [4]. Although it is not a technique a data analysis tool is obliged to offer (like decision trees), it is often a convenient alternative to other classification techniques, since it is a very efficient method and yields classifiers that are, like decision trees, easily comprehensible.

The following sections each describe one plug-in: the advanced fuzzy clustering plug-in is discussed in section 2, the decision tree plug-in in section 3, and the naive Bayes classifier plug-in in section 4. Each section first introduces the basic theory underlying the implemented methods and then describes the function blocks of the plug-in. We tried to make these sections as self-contained as possible so that any of them can be read independently of any other.

2 The Fuzzy Clustering Plug-In

The terms "classification" and "to classify" are ambiguous. With respect to classifiers like decision trees, they are used to describe the process of assigning a class from a *predefined* set to an object or case under consideration. In classical statistics, however, these terms usually have a different meaning: they are used to describe the process of dividing a dataset of sample cases into groups of similar cases, with the groups *not* predefined, but to be found by the classification algorithm. This process is also called classification, because the groups to be found are usually (and confusingly) called *classes*. To avoid the confusion that may result from this ambiguity, the latter process, i.e. dividing a dataset into groups of similar cases, is often called *clustering* or *cluster analysis*, thus replacing the ambiguous term *class* with the less ambiguous *cluster*. Nevertheless a reader should keep in mind that in this section "to classify" has a different meaning than in the following ones (except where explicitly indicated otherwise).

Cluster analysis is, as already mentioned, a technique to classify data, i.e. to divide a given dataset of sample cases into a set of classes or *clusters*. The goal

is to divide the dataset in such a way that two cases from the same cluster are as similar as possible and two cases from different clusters are as dissimilar as possible. Thus one tries to model the human ability to group similar objects or cases into classes and categories.

In classical cluster analysis [5] each case or object is assigned to exactly one cluster, i.e. classical cluster analysis yields a crisp partitioning of a dataset with "sharp" boundaries between the clusters. It is therefore also called *crisp cluster analysis*. A crisp partitioning of the dataset, however, though often indisputably successful, is not always appropriate. If the "clouds" formed by the data points corresponding to the cases or objects under consideration are not clearly separated by regions bare of any data points, but if, in contrast, in the joint domain of the attributes there are only regions of higher and lesser data point density, then the boundaries between the clusters can only be drawn with a certain amount of arbitrariness. Due to this arbitrariness it may be doubted, at least for data points close to the boundaries, whether a definite assignment to one class is justified.

An intuitive approach to deal with such situations is to make it possible that a data point belongs in part to one cluster, in part to a second etc. *Fuzzy cluster analysis* does just this: it relaxes the requirement that a data point must be assigned to exactly one cluster by allowing gradual memberships, thus offering the opportunity to deal with data points that do not belong definitely to one cluster [6, 7]. In general, the performance of fuzzy clustering algorithms is superior to that of the corresponding crisp clustering algorithms [6].

2.1 Fuzzy C-Means Algorithm

A widely used fuzzy clustering algorithm is the fuzzy c-means algorithm (*FCM*) [6] that is a built-in function of DataEngine™. This algorithm divides a given dataset $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^p$ into C clusters by minimizing the objective function

$$J(X, U, \beta) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^m d^2(\beta_i, x_j) \tag{1}$$

subject to

$$\sum_{j=1}^{n} u_{ij} > 0 \quad \text{for all } i \in \{1, \ldots, c\} \tag{2}$$

$$\sum_{i=1}^{c} u_{ij} = 1 \quad \text{for all } j \in \{1, \ldots, n\} \tag{3}$$

where $u_{ij} \in [0, 1]$ is the membership degree of datum x_j to cluster i, $\beta_i = (c_i)$ is the the prototype of cluster i, c_i is the centre of cluster i, and $d(\beta_i, x_j)$ is the distance between datum x_j and prototype β_i. The $c \times n$ matrix $U = [u_{ij}]$ is also called the fuzzy partition matrix and the parameter m is called the fuzzifier. Usually $m = 2$ is chosen.

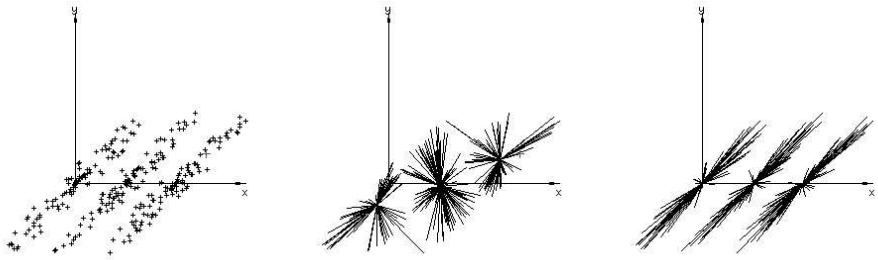

Fig. 3. Dataset 1 (left) and clustering results with the fuzzy c-means algorithm (middle) and the Gustafson-Kessel algorithm (right) for three clusters.

Constraint (2) guarantees that no cluster is empty and constraint (3) ensures that the sum of the membership degrees for each datum equals 1. Fuzzy clustering algorithms which satisfy these constraints are also called *probabilistic clustering algorithms*, since the membership degrees for one datum formally resemble the probabilities of its being a member of the different cluster.

The fuzzy c-means algorithm divides a given dataset X into c clusters of equal size and shape. The shape of the clusters depends on the distance function $d^2(c_i, x_j)$. With the Euclidean distance, the most common choice, it divides a dataset into c spherical clusters.

Although the fuzzy c-means algorithm is widely used, it fails for some classification tasks, as can be seen in Figs. 3 and 4. If the shape of the clusters is not spherical or if the clusters differ considerably in their size, the result of the fuzzy c-means algorithm is often not very intuitive and only poorly fits the data. Another problem of the fuzzy c-means algorithm is its sensitivity to noise and outliers. This sensitivity is caused by restriction (3), which equips every datum with the same weight and thus the same influence on the classification result.

To amend these problems we have implemented several fuzzy clustering algorithms that can be seen as extensions of the fuzzy c-means method. These algorithms divide a given dataset into clusters of different size and shape and are less sensitive to noise and outliers.

2.2 Possibilistic Clustering Algorithms

In applications it is a common requirement that the algorithms used should be robust. With respect to noise and outliers, this means that the performance of the algorithms should not deteriorate drastically due to noise or outliers [8]. Unfortunately, due to restriction (3) the fuzzy c-means algorithm is sensitive to noise and outliers and thus not an ideal of robustness.

One approach to reduce this sensitivity is to use an extra cluster for noise and outliers [9, 8]. Another approach is to remove restriction (3) so that it becomes possible that data that resembles noise or outliers can have a low membership

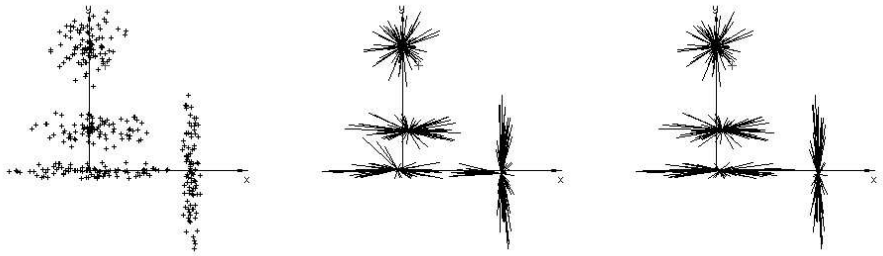

Fig. 4. Dataset 2 (left) and clustering results with the fuzzy c-means algorithm (middle) and the Gustafson-Kessel algorithm (right) for four clusters.

degree to all clusters. This approach is usually called *possibilistic clustering*. To avoid the trivial solution, i.e. $u_{ij} = 0$ for all $i \in \{1, \ldots, c\}$, $j \in \{1, \ldots, n\}$, the objective function of a possibilistic clustering algorithm has to be modified to:

$$J(X, U, \beta) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^m d^2(\beta_i, x_j) + \sum_{i=1}^{c} \eta_i \sum_{j=1}^{n} (1 - u_{ij})^m \tag{4}$$

where $\eta_i > 0$. The first term minimizes the weighted distances and the second term avoids the trivial solution. A fuzzy clustering algorithm that minimizes the above function subject to constraint (2) is called a *possibilistic clustering algorithm*, since the membership degrees for a datum resemble the possibility (in the sense of possibility theory) of its being a member of the different clusters.

Minimizing the objective function (4) with respect to the membership degrees leads to the following equation for updating the membership degrees u_{ij} [10]:

$$u_{ij} = \frac{1}{1 + \left(\frac{d^2(x_j, \beta_i)}{\eta_i} \right)^{\frac{1}{m-1}}}. \tag{5}$$

Equation (5) shows that η_i determines the distance at which the membership degree equals 0.5. If $d^2(x_j, \beta_i)$ equals η_i, the membership degree is 0.5. So it is useful to choose η_i for each cluster separately [10]. η_i can be determined, for instance, by computing the fuzzy intra cluster distance (6)

$$\eta_i = \frac{K}{N_i} \sum_{j=1}^{n} u_{ij}^m d^2(x_j, \beta_i) \tag{6}$$

where $N_i = \sum_{j=1}^{n} (u_{ij})^m$. Usually $K = 1$ is chosen. In contrast to probabilistic clustering algorithm it is recommended to choose $m = 1.5$ [11].

The fundamental difference between a probabilistic clustering algorithm and a possibilistic clustering algorithm is that a probabilistic clustering algorithm is

primarily a partitioning algorithm while a possibilistic clustering algorithm is a mode-seeking algorithm, i.e. is, a possibilistic clustering algorithm partitions a data set into c clusters, regardless of how many clusters are actually present. In contrast, each component generated by a possibilistic clustering algorithm corresponds to a dense region in the data set, i.e. if the actual number of clusters is smaller than c, some clusters might be detected twice, and if the number of clusters is higher than c, often some clusters go undetected. To avoid some clusters being detected twice while other clusters go undetected, it is recommended to initialize a possibilistic clustering algorithm with the results of the corresponding probabilistic version if the data set is not too noisy [11].

The possibilistic fuzzy c-means algorithm has been successfully used in several applications and it often helps to deal with noisy data [11, 12]. However, if the data is too noisy, the above initialization fails. In that case the user has to be very careful with the choice of η_i and the initialization of the possibilistic clustering algorithm [13, 11, 14]. As an attempt to improve the possibilistic clustering algorithm, a mixed fuzzy-possibilistic version has been suggested [14].

2.3 The Gustafson-Kessel Algorithm and the FMLE

In contrast to the fuzzy c-means algorithm the Gustafson-Kessel algorithm (GK) searches for clusters of different shape [15]. To determine the shape of the clusters the algorithm computes for each cluster a separate norm matrix A_i, $A_i = (\det C_i)^{\frac{1}{n}} C_i^{-1}$. These norm matrices are updated together with the centres of the corresponding clusters. Therefore the prototypes of the clusters are a pair (c_i, C_i), where c_i is the centre of the cluster and C_i is the covariance matrix, which defines the shapes of the clusters.

In a fashion similar to the fuzzy c-means algorithm, the Gustafson-Kessel computes the distance to the prototypes as:

$$d^2(x_j, \beta_i) = (\det C_i)^{\frac{1}{n}} (x_j - c_i)^T C_i^{-1} (x_j - c_i). \tag{7}$$

To minimize the objective function with respect to the prototypes, the prototypes are updated according to the following equations [16]:

$$c_i = \frac{1}{N_i} \sum_{j=1}^{n} u_{ij}^m x_j, \tag{8}$$

$$C_i = \frac{1}{N_i} \sum_{j=1}^{n} u_{ij}^m (x_j - c_i)(x_j - c_i)^T. \tag{9}$$

The Gustafson-Kessel algorithm is a simple fuzzy clustering algorithm to detect ellipsoidal clusters with approximately the same size but different shapes.

Another modification of the fuzzy c-means algorithm is the fuzzy maximum likelihood estimation algorithm ($FMLE$) [17]. This algorithm divides a given data set into clusters of different shape and different size. The idea of the FMLE algorithm is to interpret the data set as a p-dimensional normal distribution.

Therefore the distance of a datum to a cluster is inversely proportional to the posterior possibility that a datum x_{ij} is the realization of the ith normal distribution. Therefore the distance between a datum x_j and a cluster c_i is computed as:

$$d^2(x_j, \beta_i) = d_{ij}^2 = \frac{(det(C_i))^{\frac{1}{2}}}{P_i} \exp\left(\frac{(x_j - c_i)C_i^{-1}(x_j - c_i)^T}{2}\right). \qquad (10)$$

2.4 Simplified Versions of the GK and the FMLE

The Gustafson-Kessel and the FMLE algorithm both extend the fuzzy c-means algorithm by computing covariance matrices for each cluster. Since the covariance matrices decode norms which transform spherical clusters to ellipses or ellipsoids, the Gustafson-Kessel and the FMLE algorithm are able to detect clusters of different shape.

The idea of the simplified versions of the Gustafson-Kessel and the FMLE algorithm is to use only diagonal matrices instead of positive definite symmetric matrices as the Gustafson-Kessel or the FMLE algorithm [16], i.e. the algorithms search only for clusters that are axis parallel. This has the advantage that it is not necessary to invert matrices and to compute determinants. Therefore these algorithms have a lower computational complexity than the Gustafson-Kessel and the FMLE algorithm. These axis-parallel variants are an interesting compromise between the flexibility of the original versions and the low computational costs of the fuzzy c-means algorithm.

In addition, these algorithms are especially suited for fuzzy rule generation, i.e. for extracting descriptive rules from a data set by classifying the inputs. Since for each rule each input variable has its own interval, the clusters have to describe rectangles, which can be approximated well by axis-parallel clusters.

2.5 The DataEngine™ Plug-In

We have implemented the fuzzy clustering algorithms described in this section as a plug-in for DataEngine™. The plug-in consists of five function blocks:

Fuzzy Cluster Analysis — Several fuzzy clustering algorithms

This function block contains the main functionality of the plug-in. It contains the fuzzy clustering algorithms and executes them on the given dataset. In the configuration dialogue the algorithm, the number of clusters, and several parameters of the algorithm can be specified. Its input is unclassified data and its output is the classified data and their membership degrees to the clusters that can be used as input for other function blocks. In addition, the prototypes of the clusters are stored in a user-specified file. The prototypes can be used by other function blocks of the plug-in, for instance, as a classifier.

Classification — Classify a dataset

This function block classifies a dataset with respect to clusters that have been determined with the above function block. The cluster prototypes are read from the file to which they were saved by the first function block.

Validity Measures — Evaluation of a clustering

This function block is used to assess the quality of a computed classification. Several different validity measures can be chosen.

Parameter Extraction — Extract cluster parameters

This function block extracts the parameters of the clusters. Depending on the algorithm, the centre and the covariance matrix of each cluster are extracted.

Labelling — Labelling of a classification

With this function block labels (i.e. names) can be assigned to the clusters. This can be done automatically based on labelled data or manually.

3 The Decision Tree Plug-In

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 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 observed value 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 [18–20]. 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 3.1 we illustrate this approach using a simple (artificial) medical example.

Since the success of the induction algorithm depends heavily on the attribute selection measure used, in section 3.2 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 3.3. In addition to the induction function block, this plug-in consists of function 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).

Table 1. Patient data consisting of a set of descriptive attributes together with an effective drug (effective with respect to some unspecified disease).

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

3.1 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 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 recursion 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.

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. If we neglect 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 induce 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

Table 2. The conditional distributions of the effective drug given the sex (left), the age (middle, divided into "less than 40" and "over 40") and the blood pressure (right).

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 Pressure	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

Table 3. The second order conditional distributions of the effective drug given the blood pressure and the sex (left) and the blood pressure and the age (right, divided into "less than 40" and "over 40").

No	Blood Pressure	Sex	Drug
3	high		A
5	high		A
12	high		A
1	normal	male	A
6	normal	male	A
9	normal	male	B
2	normal	female	B
7	normal	female	B
10	normal	female	A
4	low		B
8	low		B
11	low		B

No	Blood Pressure	Age	Drug
3	high		A
5	high		A
12	high		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		B
4	low		B
8	low		B

were effective in half of the cases (thus being politically correct, i.e. here non-sexist). The patient's 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%.

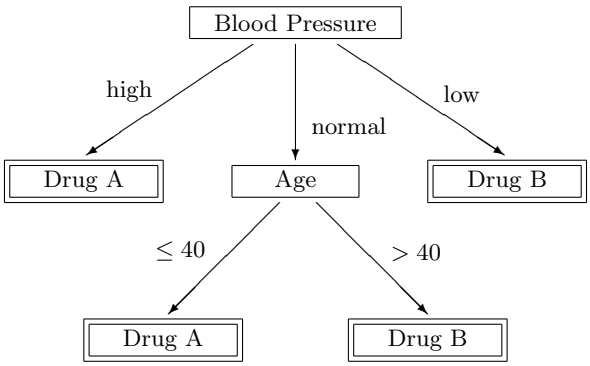

Fig. 5. The induced decision tree for the effective drug.

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. The effective drug is definite for patients with low or high blood pressure, and thus 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 sex and age (see Table 3). The patients' sex allows us to determine the correct drug for patients with normal blood pressure with a success rate of 67%. However, dividing the patients in those younger than forty and those older, perfectly separates the cases in which drug A was effective from those in which drug B was effective. Therefore the age is chosen as a second test attribute and thus a reliable method to determine the effective drug has been found. The corresponding decision tree, which can be read directly from Table 3 (right), is shown in Fig. 5.

The Induction Algorithm. The general algorithm to induce a decision tree from data is shown in Fig. 6 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 3.2.) 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 created, 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

```
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() *)
```

Fig. 6. The TDIDT (top-down induction of decision trees) algorithm.

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.

After a decision tree has been induced, it is often pruned in order to simplify it and to reduce possible overfitting to random properties of the training data. However, reasons of space prevent us from studying this step in detail.

3.2 Attribute Selection Measures

As already indicated at the beginning of this section and substantiated by the description of the general induction algorithm in the preceding section, the suc-

cess 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 led 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} (mutual information/cross entropy) [21, 22, 19]
- information gain ratio I_{gr} [19, 20]
- symmetric information gain ratio I_{sgr} [23]
- Gini index [18, 24]
- symmetric Gini index [25]
- modified Gini index [26]
- relief measure [27, 26]
- χ^2 measure
- weight of evidence [28]
- relevance [29]
- K2 metric [30, 31]
- BDeu metric [32, 31]
- minimum description length with relative frequency coding l_{rel} [28]
- minimum description length with absolute frequencies coding l_{abs} [28] (closely related to the K2 metric)
- stochastic complexity [33, 34]
- specificity gain S_{gain} [35, 36]
- (symmetric) specificity gain ratio S_{gr} [36]

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 [36] or [37].

Unfortunately, no general rule can be given as to 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.

3.3 The DataEngine™ Plug-In

We have implemented a powerful decision tree induction algorithm as a plug-in for DataEngine™ in order to improve this esteemed tool even further. This plug-in consists of four 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 be considered as unknown: in the configuration dialogue one may enter a value for the lowest known value. All values below this value are considered to be unknown. In addition the configuration dialogue 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 from which the decision tree was learned. 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 dialogue. In addition, the configuration dialogue 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 dialogue.

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 dialogue lets you enter the name of the new column and (as described for the two blocks above) a lowest known value.

xmat — compute a confusion matrix

This function block receives as input a table. Its configuration dialogue 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). These sums are the number of confusions or misclassifications, if one column contains the correct classification, the other the prediction of a classifier.

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, for example, neural networks), but you can inspect how an induced decision tree arrives at its results.

4 The Naive Bayes Classifier Plug-In

Naive Bayes classifiers [38–41] are an old and well-known type of classifiers which use a probabilistic approach to assign the classes, i.e. they try to compute the conditional probabilities of the different classes given the values of other attributes and predict the class with the highest conditional probability. Since it is usually impossible to store or even to estimate these conditional probabilities, they exploit Bayes rule and a set of conditional independence statements to simplify the task. A detailed description is given in section 4.1.

Due to the strong independence assumptions, but also because some attributes may not be able to contribute to the classification accuracy, it is not always advisable to use all available attributes. With all attributes a naive Bayes classifier is more complicated than necessary and sometimes even yields results that can be improved upon by using fewer attributes. Therefore a naive Bayes classifier should be simplified. Two very simple methods to reduce the number of attributes are discussed in section 4.2.

In section 4.3 we describe the plug-in we implemented for DataEngine™. This plug-in consists of three function blocks: one to induce (and simplify) a naive Bayes classifier, one to classify new data, and one to compute a confusion matrix to evaluate the quality of the induced classifier. The latter function block is the same as the function block *xmat* of the decision tree plug-in.

4.1 Naive Bayes Classifiers

As already mentioned above, naive Bayes classifiers use a probabilistic approach to classify data: they try to compute conditional class probabilities and then predict the most probable class. To be more precise, let C denote a class attribute with a finite domain of m classes, i.e. $\mathrm{dom}(C) = \{c_1, \ldots, c_m\}$, and let $U = \{A_1, \ldots, A_n\}$ be a set of other attributes used to describe a case or an object of the universe of discourse. These other attributes may be symbolic, i.e. $\mathrm{dom}(A_j) = \{a_1^{(j)}, \ldots, a_{m_j}^{(j)}\}$, or numeric, i.e. $\mathrm{dom}(A_j) = \mathbb{R}$. For simplicity, we always use the notation $a_{i_j}^{(j)}$ for a value of an attribute A_j, independent of whether it is a symbolic or a numeric one.[4] With this notation, a case or an object can be described by an instantiation $\omega = (a_{i_1}^{(1)}, \ldots, a_{i_n}^{(n)})$ of the attributes $A_1, \ldots, A_n$ and thus the universe of discourse is $\Omega = \mathrm{dom}(A_1) \times \ldots \times \mathrm{dom}(A_n)$.

For a given instantiation ω, a naive Bayes classifier tries to compute the conditional probability

$$P(C = c_i \mid \omega) = P\left(C = c_i \,\middle|\, \textstyle\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)}\right)$$

for all c_i and then predicts the class for which this probability is highest. Of course, it is usually impossible to store all of these conditional probabilities explicitly, so that a simple lookup would be all that is needed to find the most

[4] To be able to use this notation for numeric attributes, one simply has to choose an appropriate uncountably infinite index set $\mathcal{I}_j$, from which the index i_j is to be taken.

probable class. If there are numeric attributes, this is obvious (some parameterized function is needed then). But even if all attributes are symbolic, such an approach most often is infeasible: a class (or a class probability distribution) has to be stored for each point of the Cartesian product of the attribute domains, whose size grows exponentially with the number of attributes. To circumvent this problem, naive Bayes classifiers exploit—as their name already indicates—Bayes rule and a set of conditional independence assumptions. With Bayes rule the conditional probabilities are inverted, i.e. naive Bayes classifiers consider[5]:

$$P\left(C = c_i \,\middle|\, \textstyle\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)}\right) = \frac{f\left(\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)} \,\middle|\, C = c_i\right) \cdot P(C = c_i)}{f\left(\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)}\right)}.$$

Of course, for this inversion to be possible, the probability density function $f\big(\bigwedge_{A_j \in U} A_j = a_{i_j}^{(j)}\big)$ must be strictly positive.

There are two observations to be made about the inversion carried out above. In the first place, the denominator of the fraction on the right can be neglected, since for a given case or object to be classified, it is fixed and therefore does not have any influence on the class ranking (which is all we are interested in). In addition, its influence can always be restored by normalizing the class distribution, i.e. we can exploit:

$$f\left(\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)}\right) = \sum_{i=1}^{m} f\left(\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)} \,\middle|\, C = c_i\right) \cdot P(C = c_i).$$

It follows that we only need to consider:

$$P\left(C = c_i \,\middle|\, \textstyle\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)}\right) = \frac{P(C = c_i)}{S} f\left(\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)} \,\middle|\, C = c_i\right),$$

where S is a normalization constant.[6]

Secondly, we can see that just inverting the probabilities does not buy us anything, since the probability space is just as large as it was before the inversion. However, here the second ingredient of naive Bayes classifiers, which is responsible for the "naive" in their name, comes in, namely the conditional independence assumptions. To exploit them, we first apply the chain rule of probability:

$$\begin{aligned} &P\left(C = c_i \,\middle|\, \textstyle\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)}\right) \\ &= \frac{P(C = c_i)}{S} \prod_{k=1}^{n} f\left(A_k = a_{i_k}^{(k)} \,\middle|\, \textstyle\bigwedge_{j=1}^{k-1} A_j = a_{i_j}^{(j)}, C = c_i\right). \end{aligned}$$

[5] For simplicity, we always use a probability density function f, although this is strictly correct only, if there is at least one numeric attribute. If all attributes are symbolic, this should be a probability P. The only exception is the class attribute, since it necessarily has a finite domain.

[6] Strictly speaking, the constant S is dependent on the instantiation $(a_{i_1}^{(1)}, \ldots, a_{i_n}^{(n)})$. However, as already said above, when classifying a given case or object, this instantiation is fixed and hence we need to consider only one value S.

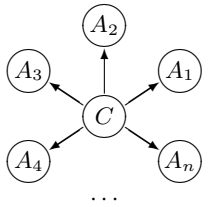

Fig. 7. A naive Bayes classifier is a Bayesian network with a star-like structure.

Now we make the crucial assumption that given the value of the class attribute, any attribute A_j is independent of any other. That is, we assume that knowing the class is enough to determine the probability (density) for a value $a_{i_j}^{(j)}$, i.e., that we need not know the values of any other attributes. Of course, this is a pretty strong assumption, which is very likely to fail. However, it considerably simplifies the formula stated above, since with it we can cancel all attributes A_j appearing in the conditions:

$$P\left(C = c_i \,\middle|\, \textstyle\bigwedge_{j=1}^{n} A_j = a_{i_j}^{(j)}\right) = \frac{P(C = c_i)}{S} \prod_{j=1}^{n} f\left(A_j = a_{i_j}^{(j)} \,\middle|\, C = c_i\right).$$

This is the fundamental formula underlying naive Bayes classifiers. For a symbolic attribute A_j the conditional probabilities $P(A_j = a_{i_j}^{(j)} \mid C = c_i)$ are stored as a simple conditional probability table. This is feasible now, since there is only one condition and hence only $m \cdot m_j$ probabilities have to be stored.[7] For numeric attributes it is usually assumed that the probability density is a Gaussian function (a normal distribution) and hence only the expected values $\mu_j(c_i)$ and the variances $\sigma_j^2(c_i)$ need to be stored in this case.

It should be noted that naive Bayes classifiers can be seen as a special type of probabilistic networks, or, to be more precise, of Bayesian networks [42]. Due to the strong independence assumptions underlying them, the corresponding network has a very simple structure: it is star-like with the class attribute being the source of all edges (see Fig. 7).

Naive Bayes classifiers can easily be induced from a dataset of preclassified sample cases. All one has to do is to estimate the conditional probabilities/probability densities $f(A_j = a_{i_j}^{(j)} \mid C = c_i)$ using, for instance, maximum likelihood estimation. For symbolic attributes, this yields:

$$\hat{P}(A_j = a_{i_j}^{(j)} \mid C = c_i) = \frac{\#(A_j = a_{i_j}^{(j)}, C = c_i)}{\#(C = c_i)},$$

where $\#(C = c_i)$ is the number of sample cases that belong to the class c_i and $\#(A_j = a_{i_j}^{(j)}, C = c_i)$ is the number of sample cases belonging to class c_i and

[7] Actually only $m \cdot (m_j - 1)$ probabilities are really necessary. Since the probabilities have to add up to one, one value can be discarded from each conditional distribution. However, in implementations it is usually much easier to store all probabilities.

Table 4. A naive Bayes classifier for the iris data. The normal distributions are described by stating $\hat{\mu} \pm \hat{\sigma}$.

iris type	setosa	versicolor	virginica
prior prob.	0.333	0.333	0.333
petal length	1.46 ± 0.17	4.26 ± 0.46	5.55 ± 0.55
petal width	0.24 ± 0.11	1.33 ± 0.20	2.03 ± 0.27

having the value $a_{i_j}^{(j)}$ for the attribute A_j. To ensure that the probability is strictly positive (see above), it is assumed that there is at least one example for each class in the dataset. Otherwise the class is simply removed from the domain of the class attribute. If an attribute value does not occur given some class, its probability is either set to $\frac{1}{2N}$, where N is the number of sample cases, or a uniform prior of, for example, $\frac{1}{N}$ is always added to the estimated distribution, which is then renormalized (Laplace correction).

For a numeric attribute A_j the standard maximum likelihood estimation functions

$$\hat{\mu}_j(c_i) = \frac{1}{\#(C = c_i)} \sum_{k=1}^{\#(C=c_i)} a_{i_j(k)}^{(j)}$$

for the expected value, where $a_{i_j(k)}^{(j)}$ is the value of the attribute A_j in the k-th sample case belonging to class c_i, and

$$\hat{\sigma}_j^2(c_i) = \frac{1}{\#(C = c_i)} \sum_{k=1}^{\#(C=c_i)} \left(a_{i_j(k)}^{(j)} - \hat{\mu}_j(c_i)\right)^2$$

for the variance can be used.

As an illustrative example, let us take a look at the well-known iris data [43]. The problem is to predict the iris type (iris setosa, iris versicolor, or iris virginica) from measurements of the sepal length and width and the petal length and width. Due to the limited number of dimensions of a sheet of paper we confine ourselves to the latter two measures. The naive Bayes classifier induced from these two measures and all 150 cases is shown in Table 4. It is easy to see from this table how different petal lengths and widths provide evidence for the different types of iris flowers. The conditional probability density functions used by this naive Bayes classifier to predict the iris type are shown graphically in Fig. 8. The ellipses are the 2σ-boundaries of the (bivariate) normal distributions. As a consequence of the strong conditional independence assumptions, these ellipses are axis-parallel: the normal distributions are estimated separately for each dimension and no covariance is taken into account.

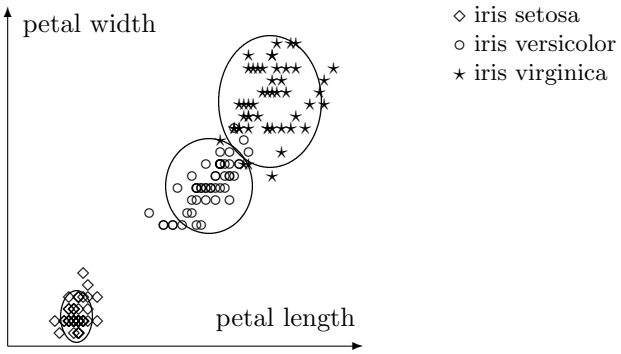

Fig. 8. Naive Bayes density functions for the iris data. The ellipses are the 2σ-boundaries of the probability density functions.

4.2 Classifier Simplification

A naive Bayes classifier makes strong independence assumptions (see above). It is not surprising that these assumptions are likely to fail. If they fail, the classifier may be worse than necessary. In addition, some attributes may not contribute to the classification accuracy, making the classifier more complicated than necessary. To cope with these problems, simplification methods may be used, for instance, simple greedy attribute selection. With this procedure one can hope to find a subset of attributes for which the strong assumptions hold at least approximately.

We consider here two very simple, but effective, attribute selection methods: the first method starts with a classifier that simply predicts the majority class and does not use any attribute information. Then attributes are added one by one. In each step that attribute is selected which, if added, leads to the smallest number of misclassifications on the training data. The process stops when adding any of the remaining attributes does not reduce the number of errors.

The second method is a reversal of the first. It starts with a classifier that uses all available attributes and then removes attributes step by step. In each step that attribute is selected which, if removed, leads to the smallest number of misclassifications on the training data. The process stops when removing any of the remaining attributes leads to a larger number of errors.

4.3 The DataEngine™ Plug-In

We have implemented a naive Bayes classifier as a plug-in DataEngine™ in order to improve the capabilities of this tool. It consists of three function blocks:

nbi — naive Bayes classifier induction

This function block receives as input a table of classified sample cases and induces a naive Bayes classifier. 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, as the decision tree function blocks, a facility to specify which table fields should be considered as unknown: in the configuration dialogue you may enter a value for the lowest known value. All values below this value are considered to be unknown. In addition, the configuration dialogue lets you choose a simplification method and you can specify the Laplace correction to be used (as a multiple of the standard value $\frac{1}{n}$, where n is the number of tuples from which the classifier is induced) and the name of a file into which the induced naive Bayes classifier should be saved.

nbc — naive Bayes classification
This function block receives as input an induced naive Bayes classifier stored in a file and a table of cases. It executes the naive Bayes classifier for each tuple in the table and adds to it a new column containing the class predicted by the classifier. The configuration dialogue lets you enter the name of the classification column, the Laplace correction to be used (again as a multiple of the standard value $\frac{1}{n}$, where n is the number of tuples from which the classifier was induced), and (just as described for the block above) a lowest known value. In addition you can request an additional column into which a confidence value is written for each classified tuple. This confidence value is the probability of the predicted class as computed by the classifier.

xmat — compute a confusion matrix
This function blocks is identical to the one with the same name in the decision tree plug-in, see section 3.3.

All function block dealing directly with naive Bayes classifiers (*nbi* and *nbc*) also comprise a viewer which lets you inspect the constructed naive Bayes classifier 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 classifier as a black box (as is usually the case for, for example, neural networks), but you can inspect what evidence it exploits to arrive at its results.

5 Conclusions

None of the data analysis tools available today is perfect, but with some tools a user has to be content with the capabilities supplied by the programmers, without a chance of ever overcoming the restrictions and weaknesses of the shipped version. Some, though very few tools, however, with DataEngine™ among them, offer the possibility for user-specific extensions. In this paper we described how we exploited the DataEngine™ interface for user-defined function blocks to implement three plug-ins that overcome weaknesses of the original product and

extend its capabilities. The fact that two of the plug-ins have become commercial products that are sold now by the makers of DataEngine™ as add-ons to their basic tool shows that it can pay for a software company to produce open and extensible systems, although it takes considerable effort to do so.

References

1. Nürnberger A. and Timm H.: 'OR Software: DataEngine'. OR Spektrum, Vol. 21, pp. 305–313 (1999).
2. Timm H.: 'A fuzzy cluster analysis plug-in for dataengine'. In 'Proc. 2nd Data Analysis Symposium' (Aachen) (1998).
3. Borgelt C.: 'A decision tree plug-in for dataengine'. In 'Proc. 2nd Data Analysis Symposium' (Aachen) (1998).
4. Borgelt C.: 'A naive bayes classifier plug-in for dataengine'. In 'Proc. 2nd Data Analysis Symposium' (Aachen) (1999).
5. Berry M. and Linoff G.: 'Data Mining Techniques — For Marketing, Sales and Customer Support'. Chichester, England, J. Wiley & Sons (1997).
6. Bezdek J.: 'Pattern Recognition with Fuzzy Objective Function Algorithms'. New York, NY, Plenum (1981).
7. Bezdek J. and Pal S.: 'Fuzzy Models for Pattern Recognition — Methods that Search for Structures in Data'. Piscataway, NJ, IEEE Press (1992).
8. Davé R. and Krishnapuram R.: 'Robust clustering methods: A unified view'. IEEE Trans. on Fuzzy Systems, Vol. 5, pp. 270–293 (1997).
9. Davé R.: 'Characterization and detection of noise in clustering'. Pattern Recognition Letters, Vol. 12, pp. 657–664 (1991).
10. Krishnapuram R. and Keller J.: 'A possibilistic approach to clustering'. IEEE Transactions on Fuzzy Systems, Vol. 1, pp. 98–110 (1993).
11. Krishnapuram R. and Keller J.: 'The possibilistic c-means algorithm: Insights and recommendations'. IEEE Trans. on Fuzzy Systems, Vol. 4, pp. 385–393 (1996).
12. Nasroui O. and Krishnapuram R.: 'Crisp interpretations of fuzzy and possibilistic clustering algorithm'. In 'Proc. 3rd European Congress on Fuzzy and Intelligent Technologies (EUFIT'95, Aachen, Germany)' (Aachen: Verlag Mainz), pp. 1312–1318 (1994).
13. Barni M., Capellini V. and Mecocci A.: 'Comments on "a possibilistic approach to clustering"'. IEEE Transactions on Fuzzy Systems, Vol. 4, pp. 393–396 (1996).
14. Pal N., Pal K. and Bezdek J.: 'A mixed c-means clustering model'. In 'Proc. 6th IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'97, Barcelona, Spain)' (Piscataway, NJ: IEEE Press), pp. 11–21 (1997).
15. Gustafson E. and Kessel W.: 'Fuzzy clustering with a fuzzy covariance matrix'. In 'Proc. IEEE Conf. on Decision and Control (CDC'79, San Diego, CA)' (Piscataway, NJ: IEEE Press), pp. 761–766 (1979).
16. Höppner F., Klawonn F., Kruse R. and Runkler T.: 'Fuzzy Cluster Analysis'. Chichester, J. Wiley & Sons (1999).
17. Gath I. and Geva A.: 'Unsupervised optimal fuzzy clustering'. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), Vol. 11, pp. 773–781 (1989).
18. Breiman L., Friedman J., Olshen R. and Stone C.: 'Classification and Regression Trees'. Belmont, CA, Wadsworth International (1984).
19. Quinlan J.: 'Induction of decision trees'. Machine Learning, Vol. 1, pp. 81–106 (1986).

20. Quinlan J.: 'C4.5: Programs for Machine Learning'. San Mateo, CA, Morgan Kaufman (1993).
21. Kullback S. and Leibler R.: 'On information and sufficiency'. Ann. Math. Statistics, Vol. 22, pp. 79–86 (1951).
22. Chow C. and Liu C.: 'Approximating discrete probability distributions with dependence trees'. IEEE Trans. on Information Theory, Vol. 14(3), pp. 462–467 (1968).
23. Lopez de Mantaras R.: 'A distance-based attribute selection measure for decision tree induction'. Machine Learning, Vol. 6, pp. 81–92 (1991).
24. Wehenkel L.: 'On uncertainty measures used for decision tree induction'. In 'Proc. Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'96)' (Granada, Spain), pp. 413–417 (1996).
25. Zhou X. and Dillon T.: 'A statistical-heuristic feature selection criterion for decision tree induction'. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), Vol. 13, pp. 834–841 (1991).
26. Kononenko I.: 'Estimating attributes: Analysis and extensions of relief'. In 'Proc. 7th Europ. Conf. on Machine Learning (ECML'94)' (New York, NY: Springer) (1994).
27. Kira K. and Rendell L.: 'A practical approach to feature selection'. In 'Proc. 9th Int. Conf. on Machine Learning (ICML'92)' (San Franscisco, CA: Morgan Kaufman), pp. 250–256 (1992).
28. Kononenko I.: 'On biases in estimating multi-valued attributes'. In 'Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining (KDD'95, Montreal, Canada)' (Menlo Park, CA: AAAI Press), pp. 1034–1040 (1995).
29. Baim P.: 'A method for attribute selection in inductive learning systems'. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), Vol. 10, pp. 888–896 (1988).
30. Cooper G. and Herskovits E.: 'A Bayesian Method for the Induction of Probabilistic Networks from Data'. Machine Learning. Dordrecht, Kluwer (1992).
31. Heckerman D., Geiger D. and Chickering D.: 'Learning bayesian networks: The combination of knowledge and statistical data'. Machine Learning, Vol. 20, pp. 197–243 (1995).
32. Buntine W.: 'Theory refinement on bayesian networks'. In 'Proc. 7th Conf. on Uncertainty in Artificial Intelligence' (Los Angeles, CA: Morgan Kaufman), pp. 52–60 (1991).
33. Krichevsky R. and Trofimov V.: 'The performance of universal coding'. IEEE Trans. on Information Theory, Vol. 27(2), pp. 199–207 (1983).
34. Rissanen J.: 'Stochastic complexity'. Journal of the Royal Statistical Society (Series B), Vol. 49, pp. 223–239 (1987).
35. Gebhardt J. and Kruse R.: 'Tightest hypertree decompositions of multivariate possibility distributions'. In 'Proc. Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'96)' (Granada, Spain), pp. 923–927 (1996).
36. Borgelt C. and Kruse R.: 'Evaluation measures for learning probabilistic and possibilistic networks'. In 'Proc. 6th IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'97, Barcelona, Spain)' (Piscataway, NJ: IEEE Press), pp. 1034–1038 (1997).
37. Borgelt C. and Kruse R.: 'Attributauswahlmaße für die induktion von entscheidungsbäumen: Ein überblick'. In 'Data Mining: Theoretische Aspekte und Anwendungen', G. Nakhaeizadeh, Ed. Physica-Verlag, Heidelberg, pp. 77–98 (1998).
38. Good I.: 'The Estimation of Probabilities: An Essay on Modern Bayesian Methods'. Cambridge, MA, MIT Press (1965).

39. Duda R. and Hart P.: 'Pattern Classification and Scene Analysis'. New York, NY, J. Wiley & Sons (1973).
40. Langley P., Iba W. and Thompson K.: 'An analysis of bayesian classifiers'. In 'Proc. 10th Nat. Conf. on Artificial Intelligence (AAAI'92, San Jose, CA, USA)' (Menlo Park and Cambridge, CA: AAAI Press and MIT Press), pp. 223–228 (1992).
41. Langley P. and Sage S.: 'Induction of selective bayesian classifiers'. In 'Proc. 10th Conf. on Uncertainty in Artificial Intelligence (UAI'94, Seattle, WA, USA)' (San Mateo, CA: Morgan Kaufmann), pp. 399–406 (1994).
42. Pearl J.: 'Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (2nd edition)'. San Mateo, CA, Morgan Kaufman (1992).
43. Fisher R.: 'The use of multiple measurements in taxonomic problems'. Annals of Eugenics, Vol. 7(2), pp. 179–188 (1936).