Chapter 2
Techniques of Decision Tree Induction

Finding optimal DT for given data is not easy (with exceptions of some trivial cases). The hierarchical structure of DT models could suggest that the optimization process is also nicely reduced with subsequent splits, but it is not so. It is important to realize that optimization of a criterion for tree node is not the same as optimization of the whole tree or even subtree. The difference has been proved formally by de Sá (2001) for decision trees of some specific, simple form—where each class is represented by one leaf. On this assumption, each tree node can be assigned a subset of class labels in such a way that the root node is assigned the full set of class labels, and subsequent splits divide the set of labels assigned to the node being split into disjoint parts. Further assumption of independence of the features used along each path of the tree leads to the conclusion that probability of correct classification to class c_k is given by

$$P_C(c_k) = \prod_{i_k=1}^{n_k-1} P_C(c_k|N_{i_k}), \qquad (2.1)$$

where $(N_1, \ldots, N_{n_k})$ is the tree branch ended with the leaf assigned c_k label and $P_C(c_k|N_i)$ is the probability of correct decision at node N_i. Then, the probability of correct classification by the whole tree is

$$P_C(T) = \sum_{c_k \in \mathscr{C}} P(c_k) \prod_{i_k=1}^{n_k-1} P_C(c_k|N_{i_k}). \qquad (2.2)$$

On the other hand, the probability of correct classification at node N is a linear combination of probabilities of correct classification of objects belonging to subsequent classes:

$$P_C(N) = \frac{\sum_{c_k \in C(N)} P(k) P_C(c_k|N)}{\sum_{c_k \in C(N)} P(k)}, \qquad (2.3)$$

K. Grąbczewski, *Meta-Learning in Decision Tree Induction*,
Studies in Computational Intelligence 498, DOI: 10.1007/978-3-319-00960-5_2,
© Springer International Publishing Switzerland 2014

where $C(N)$ is the set of class labels assigned to node N. As a result, optimization of this probability can not be equivalent to optimization of $P_C(T)$, because the latter depends nonlinearly on subsequent $P_C(c_k|N)$.

With more general definition of DTs (for example, the one presented in Sect. 1.2), the dependencies of tree accuracy on node accuracies is yet less straightforward, so optimization at each tree node can also be quite different from the optimization of the whole tree. It means that to find a global optimum, one would have to examine all possible trees, but even in simple (but not trivial) cases, it is not possible, because the number of possible trees is usually infinite (or at least very large, as it grows exponentially with target tree depth). Therefore, in practical DT induction, the trees are constructed with heuristic search methods. Since classification accuracy at tree nodes does not directly affect the accuracy of the whole tree, instead of the accuracy criterion, other measures of split quality are used as heuristics and turn out to be more valuable.

Practical approaches put some constraints on the domain (assume some form of node split functions) and perform a search based optimization in such limited space of models. Scientists working in the area have come up with large number of different ideas of how to restrict the search and how to optimize model selection. It is not possible to list them all even in a book, but it is possible to present the most interesting contributions to the field, proposed in recent decades. It would not make much sense to present many similar solutions, so this chapter reviews the most popular kinds of decision tree algorithms and some assessed as very interesting or possibly valuable. Of course, the selection is subjective, but it definitely can give a good grasp of the field, even to a novice to machine learning, and good orientation in the area to people who do not have everyday experience in this branch of computational intelligence.

2.1 Recursive Top-Down Splits

The most common approaches to decision tree induction are based on recursive top-down splits of the training dataset. Given a method to split DT nodes, or more precisely, to split the training data corresponding to the node, it is executed at each tree node to find the best splits. Denoting such split method by *BestSplit()*, the recursive DT construction algorithm can be formally written as Algorithm 2.1. Most often, splits are found with an exhaustive search through the collection of all possible splits of the node at hand. In such approaches, the *BestSplit* function analyzes each candidate split with a *split quality measure* (SQM) provided as one of the most significant configuration parameters of the method. Many split quality measures are based on common idea of *impurity reduction* (or *purity gain*). Provided a measure of data sample purity (homogeneity), split quality may be estimated as the increase of the homogeneity between the tree nodes after the split and before the split. *Impurity measures* (or *criteria* or *indices*) should satisfy some conditions to be compatible with the idea: a node containing data objects representing one class only should get minimum possible value of the index (usually zero). Maximum values should be given to maximally mixed samples (all classes represented by the same number of objects).

Algorithm 2.1 (Common DT induction approach)

Prototype: CommonDTRec(D,BestSplit)
Input: *Training dataset D, node splitting procedure BestSplit.*
Output: *Decision tree (= the root node of the tree).*
The algorithm:

```
1. $s \leftarrow BestSplit(D)$
2. if $s \neq \perp$ then /* a split has been returned */
      a. $\{D_1, \ldots, D_n\} \leftarrow s(D)$ (split node $N$)
      b. for $i = 1, \ldots, n$ do
            $N_i \leftarrow CommonDTRec(D_i, BestSplit)$
      c. $Children \leftarrow (N_1, \ldots, N_n)$
   else
     $Children \leftarrow \perp$
3. return $(D, s, Children)$
```

Denoting the purity criterion as I, we get the following formula of *purity gain* (*impurity reduction*):

$$\Delta I(s, D) = I(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} I(D_i) \tag{2.4}$$

where s is the split, D is the dataset to be split and $s(D) = (D_1, \ldots, D_k)$. Given the index I, the best splits of dataset D are those maximizing $\Delta I(s, D)$.

Exhaustive search for best split guarantees local maximum, however can be costly. To avoid the cost, the exhaustive search is sometimes replaced by statistical tests and/or discrimination methods. Statistical tests can determine the features which seem to maximize the probability of providing attractive data splits. Discrimination methods can calculate split points without the necessity of checking all possible splits. Separating feature selection from split determination may significantly reduce computational complexity of a single split procedure, but it is important to realize that it does not necessarily imply faster tree construction (the resulting trees may be larger, so more splits may compose the final trees).

On the other hand, yet more complex search methods can be run, to examine split advantages more thoroughly. For example, one can estimate split quality on the basis of analysis of potential further splits. More details on DT search procedures can be found in Sect. 3.1.1.

Split criteria and search methods are not all the differences between DT induction algorithms. Some methods use only binary splits, while others accept splits into more than two subnodes. In some trees, only univariate splits are allowed and others perform multivariate analyses. Many more detailed differences could be enumerated. The following sections present many DT induction algorithms in many interesting aspects. Some alternative collections of decision tree induction techniques can be found for example in Murthy (1998), Rokach and Maimon (2008, 2010), Kotsiantis (2011). More organized analysis and a unified model of DT induction methods is the subject of Chap. 3.

2.2 Univariate Decision Trees

Numerous DT construction algorithms result in models where splits are performed on the basis of simple conditions concerning single features. Decision borders of such models are perpendicular to axes of the space of data object descriptions. From one point of view, it is a serious limitation of the approaches, but from another, decisions can be described with readable formulae making the models 1comprehensible and thanks to that easier acceptable by experts in many fields, for example in medicine, where a responsible clinician can accept support from an artificial decision system only if it provides comprehensible descriptions of its suggestions.

2.2.1 ID3

Iterative Dichotomiser 3 (ID3) (Quinlan 1986; Mitchell 1997) is one of the earliest ideas of DT induction. Its split criterion was founded on information theory. The most serious drawback of ID3 is the requirement that the data description may include only discrete features. When the original data table contains numeric features, they must be first discretized. Success of data mining processes consisting of data discretization and final model creation usually depends on the former part more than on the latter. Therefore, estimation of the efficiency and accuracy of ID3 in application to continuous data does not make much sense, because with one discretization method the results may be very good and with another one—completely wrong.

The method is a typical example of top-down recursive induction presented in Algorithm 2.1. Split qualities are estimated with the purity gain criterion (2.4) using entropy as node impurity measure:

$$I_E(D) = H_C(D) = -\sum_{c\in\mathscr{C}} P(c|D)\log_2 P(c|D). \tag{2.5}$$

Such combination of formulae (entropy reduction) is called *information gain* (IG) criterion:

$$IG(s, D) = \Delta I_E(s, D). \tag{2.6}$$

In practice, the probabilities of classes within the node N are usually estimated by ratios $\frac{n_c}{n}$ of the numbers of objects in node N data representing class c and the numbers of all objects falling into N. When implementing the information gain criterion for the sake of DT induction, one usually simplifies the formulae. Converting expressions according to the equality

$$-\sum_{c\in\mathscr{C}} \frac{n_c}{n} \log \frac{n_c}{n} = \log n - \frac{1}{n} \sum_{c\in\mathscr{C}} n_c \log n_c,$$

the information gain resulting from the split of N into parts N^p can be written as

$$IG = \log n + \frac{1}{n}\left(-\sum_c n_c \log n_c - \sum_p n^p \log n^p + \sum_p \sum_c n_c^p \log n_c^p \right). \quad (2.7)$$

Because in DT induction we are interested in comparison between splits, not in precise calculation of IG, the constant parts of the formula given above can be ignored, and only the second and third components in the big parentheses need to be calculated.

In each step of ID3 algorithm, a node is split into as many subnodes as the number of possible values of the feature used for the split. The exhaustive search for best split, in this case, just estimates the quality of each feature, because only one split is possible per feature. The feature offering maximal entropy reduction is selected, the node split, and the feature used for the split is removed from the data passed down to the subnodes, because it is no longer useful in the tree branch (all objects have the same value of this feature).

An important disadvantage of such split technique is that the features with many possible values are preferred over those with small counts of symbols, even when the former are not too valuable—in an extreme case, if each object has a unique value of a feature, the feature will reduce the entropy to zero, so will be treated as very precious, while in fact, its value is overestimated due to the split into many nodes with single data objects.

Apart from the main ID3 algorithm, Quinlan (1986) has presented some other interesting contributions. One of them is the method for fastening DT induction, when training dataset is very large. Quinlan proposed an iterative framework discussed also by O'Keefe (1983). It is based on using a *window*—a subset of the training dataset instead of all training objects. In such approach, ID3 may need a number of tree induction iterations to provide final classification tree. The process starts with building a tree to classify objects in the window with maximum accuracy. Then, the tree classifies the objects outside the window. If all the objects are classified correctly, the tree is the final result. Otherwise, a selection of incorrectly classified objects is added to the window and next tree is generated. If the window is allowed to grow to the size capable of containing all training data objects, the process is guaranteed to end up with a maximally accurate tree (with respect to the training data). If not, some problems with convergence may also occur.

Quinlan (1986) has also proposed some solutions to avoid overfitting noisy data. Stop criterion defined as zero information gain is too little to guarantee generalization of tree models. Nonzero information gain can very often be observed even in completely random distributions. A threshold for information gain is not recommended either, because finding proper threshold value can be difficult. Therefore Quinlan (1986) proposed a method based on Pearson's χ^2-test for stochastic independence, described in appendix Sect. A.2.2 For example, when an attribute has v possible values $a_1, \ldots, a_v$ and classification problem defines k classes $c_1, \ldots, c_k$, then the independence between the attribute and the classes can be estimated with the Pearson's χ^2 statistic calculated for the contingency table reflecting the joint distribution of the two variables. The statistic may be confronted with the χ^2 distribution with $(v-1)(k-1)$ degrees of freedom and a given confidence level, to verify whether the attribute is irrelevant. When the hypothesis about the independence can not be rejected, the attribute should not define the next split in the tree—a leaf should be generated regardless of its impurity.

Quinlan (1986) mentioned several possible ways of dealing with missing values (mostly proposed earlier by other authors) like imputing the most probable value, using fractional objects, predicting the value with a decision tree trained for this purpose, and others.

2.2.2 CART

Classification and Regression Trees (CART) is one of the most popular and very successful methods of DT induction (Breiman et al. 1984; Michie et al. 1994; Cherkassky and Mulier 1998). The algorithm is nonparametric and creates binary trees from data described by both continuous and discrete features. For continuous features, all possible binary splits into intervals $(-\infty, a]$ and (a, ∞) are considered. For discrete attributes, the analysis concerns all possible splits of the set of symbols into two disjoint and complementary subsets.

Exhaustive search for the best splits estimates split qualities with the impurity reduction criterion (2.4) with impurity defined as so called *Gini (diversity) index*:

$$I_G(D) = 1 - \sum_{c \in \mathscr{C}} P(c|D)^2. \tag{2.8}$$

In place of Gini index, it is also possible to use entropy (2.5) or any other measure of impurity.

Breiman et al. (1984) also proposed a technique named *twoing*, which was created to handle multi-class problems by two-class criteria. Twoing means grouping the classes into two superclasses and performing two-class analysis for the groups, instead of the original classes. Naturally, when the number of classes is large, the number of possible groupings can cause combinatorial explosion, if one tries to check all possibilities. Instead of analyzing all splits for all possible class groupings, Breiman et al. (1984) proposed an efficient procedure to determine optimal superclasses for each possible split. The procedure is valid for two-class impurity criterion (compatible with Gini index) defined as

$$I(D) = P(c_1|D)P(c_2|D). \tag{2.9}$$

Breiman et al. (1984) proved that for given binary split s, which for a dataset D generates subsets D_L and D_R, maximum decrease of impurity is obtained when one superclass contains all the classes c for which $P(c|D_L) \geq P(c|D_R)$, and the second superclass—the remaining classes. This result lets keep attractive computational complexity of the twoing procedure.

In CART, each tree node is assigned the class label dominating within the node. There is also a possibility to respect misclassification costs in the decisions.

Missing data values are handled with a technique of *surrogate splits*. When a data object is not described with a value necessary for the test of a tree node, it is passed to another test, exploiting another feature to generate a split, as similar to the one of the original test as possible. Several surrogate splits can be found and used for data with missing values in appropriate order.

As indicated in the name of CART, the method is designed to be applicable to both classification and regression problems. Approximation trees are very similar to the classification ones, but instead of class labels, the nodes are assigned some real values. Such tree definition allows the trees to represent piecewise constant functions, so to approximate less trivial functions with low mean squared error, the trees need to be large.

Many improvements and extensions to CART solutions have been proposed later. For example Strobl et al. (2005) proposed using p-values to obtain unbiased feature selection (see Sect. 2.7) and Piccarreta (2008) extended Gini criterion to ordinal response variables.

Cost-Complexity Optimization

Breiman et al. (1984) have also proposed an interesting method for pruning CART after learning, as stop criteria are neither as flexible nor accurate as post-pruning methods. Although CART offers a stop criterion in the form of minimum node size specification (minimum number of training data falling into the node), the main tool for adjusting tree size to given problem is the *cost-complexity optimization*. As suggested by the name, instead of just training error minimization, which usually leads to overfitting the training data, the optimization concerns a risk measure involving both misclassification cost and size (complexity) of the model, defined as the number of leaves of the tree (see Eq. (2.79) and Sect. 2.4.3.2 for detailed explanation). A parameter α controls the trade-off between training data reclassification accuracy and size of the tree. To determine the optimal value of α, validation procedures are proposed to estimate the performance of the candidate values and select the best one. The validation can be performed on the basis of a separate validation dataset or by cross-validation. In the case of CV, in each pass, a DT is built and analyzed to determine all the threshold αs, that is, all the values of the parameter for which a node obtains the same combined cost as the subtree and reduced to a leaf, so that for α less than the threshold it is advantageous to leave the subtree as it is, and for values greater than the threshold—to replace the subtree by a leaf. Finally, a tree is built for the whole dataset, and its threshold αs are determined. On the basis of the CV, for each threshold, its average risk value is calculated and the one providing minimum risk is chosen as the optimum. The final tree is pruned in the way that minimizes the risk for the selected threshold α. For more formal presentation of the algorithm see Sect. 2.4.3.2.

Pruning Control with Standard Error Margin

CART implementation of the validation procedure introduced a parameter to enforce simpler trees than resulting from normal cost-complexity analysis. As a result the cost-complexity optimization usually comes in two versions: 1SE and 0SE. The acronym SE stands for "standard error". 0SE denotes just the fundamental version of the method (without standard error based correction) while 1SE signifies the modified version, where standard error is estimated and model selection prefers simpler trees with the reservation that the cost can not increase by more than the value of the standard error. More information on the SE parameter and methods of its calculation can be found in Sect. 3.2.4.1.

2.2.3 C4.5

Another very popular DT induction system (next to CART) is C4.5 by Quinlan (1993). It has found numerous applications. The system arose from ID3 and shares many solutions with its ancestor. Main differences introduced in C4.5 are:

- modified node impurity measure,
- support for direct handling continuous attributes (no necessity to discretize them),
- introduction of a pruning method,
- precise methods for handling data with missing values.

Impurity Measure

Modified measure of node impurity aimed at eliminating bias in split feature selection, that is, favoring features with many possible values by the information gain criterion used in ID3. To replace the IG, Quinlan (1993) proposed *information gain ratio* (IGR) defined as

$$\Delta I(s, D) = \frac{\Delta I_E(s, D)}{SI(s, D)}, \tag{2.10}$$

where split information $SI(s, D)$ is the entropy of the split $s(D) = (D_1, \ldots, D_n)$:

$$SI(s, D) = -\sum_i p_i \log_2 p_i, \quad \left(p_i = \frac{|D_i|}{|D|}\right). \tag{2.11}$$

Handling Continuous Attributes

The support for continuous attributes in the data is organized similarly to CART. All sensible binary splits, deduced from the training data, are examined and the one with the best score (here the largest information gain ratio) chosen. Unlike symbolic features, continuous attributes may occur at different levels of the same tree many times (symbolic ones, when used in the tree, are no longer useful and because of that are not considered in further splits in the branch).

DT Pruning

In ID3 a statistical test of independence served as a stop criterion to prevent oversized trees, overfitting the training data. C4.5 offers another technique of generalization control. It builds (almost) maximally accurate trees and then prunes them to get rid of too detailed nodes that have not learned any general classification rule but just adjusted to specific data objects present in the training sample. The word "almost" added in parenthesis reflects what can be found in the source code of C4.5 about the process of tree construction: C4.5 has a parameter MINOBJS, which controls a pre-pruning method. If a node to be split contains too small number of objects or the split would generate too small nodes, further splits are rejected. After the tree is constructed, each node is tested with a statistical tool to estimate the probability that the node split causes error reduction (assuming binomial distribution of erroneous decisions). Each node, for which the probability is below a given threshold, is pruned or the subtree rooted in the node is replaced by its best subtree (the technique was named *grafting*). More details about C4.5 pre-pruning and post-pruning (*Error-Based Pruning*) methods can be found in Sect. 2.4.2.2.

Handling Missing Values

Objects with missing values can also be used in both the process of C4.5 DT construction and in further classification with a ready tree. At the stage of tree construction, in calculation of IGR, the objects with missing values of the feature being analyzed are ignored—the index is computed for a reduced set of objects and the result is scaled by the factor of probability of value accessibility (estimated by the fraction of the number of objects with non-missing value of the feature and the number of all training objects at the node). When the training data sample is split for subnodes creation, weights are introduced to reflect that it is not certain which path should be followed by the training data objects with missing decision feature values. The weights may be interpreted as the probabilities of meeting or not the condition assigned to the node. They are calculated as the proportions reflecting the distribution of other data (with non-missing value) among the subnodes. When the weights are introduced, they must be considered also in further calculations of the IGR—wherever cardinalities are used (see Eqs. (2.4), (2.10) and (2.11)), sums of the weights are calculated instead of just the numbers of elements (naturally, the default initial weight value for each object is 1). Similarly, at classification stage, if a decision feature value is missing for a data object, all subnodes are tested and decisions obtained from each path are combined by adequate weighting to obtain final probabilities of the classes for the object.

Other Interesting Solutions

Apart from the decision tree algorithm, C4.5 system offers a methodology for building classifiers based on sets of logical rules. The algorithm called "C4.5 rules" starts with building a decision tree and converting it into compatible classification rules, but then the rules are subject to a simplification (pruning) process, which can significantly change the decision function of the model. Rules pruning is performed by removing premises if without them reclassification does not get deteriorated. Each rule is simplified separately, so the resulting rule set based classifier may be significantly different than the original tree (in practice, usually less accurate).

A modified version of C4.5, named *C5.0* or *See5*, is a commercial product and its popularity is very low in comparison to the ubiquitous C4.5. Because of that, also its results are not so commonly known as those of C4.5.

2.2.4 Cal5

Müller and Wysotzki (1994, 1997) have created a decision tree induction algorithm *Cal5* for classification of objects described in spaces of continuous features. Fundamental element of the method is its procedure dividing continuous features into intervals with application of statistical tools to estimate tree node purity.

As most other DT induction algorithms, Cal5 recursively splits nodes into subnodes, starting with the root node containing the whole training data sample. Analysis of each node consists of three main steps:

- selection of the best attribute for the split,
- discretization of the attribute (dividing it into intervals),
- merging adjacent intervals resulting from the discretization.

Algorithm 2.2 sketches the topmost procedure of the method. The parameters mentioned in the input specification section control some details of the three main steps of each node analysis:

- the selection of a measure to estimate attribute eligibility for the split (Q or IG),
- threshold s, defining minimum probability of correct classification by given node, that makes it a leaf,
- confidence level α for statistical tests performed within the discretization process.

Algorithm 2.2 (DT induction by Cal5)

Prototype:* *Cal5(D)
Input: *Training dataset D, some configuration parameters.*
Output: *Decision tree.*
The algorithm:

```
1. $A \leftarrow BestAttribute(D)$
2. $Intervals \leftarrow Discretize(A, D)$
3. $Intervals \leftarrow MergeIfReasonable(Intervals)$
4. If $|Intervals| > 1$
      a. For $i = 1, \ldots, |Intervals|$
            $N_i \leftarrow Cal5(Intervals_i)$
      b. $Children \leftarrow (N_1, \ldots, N_{|Intervals|})$
   else
      $Children = \perp$
5. return $(D, s_{Intervals}, Children)$
```

Precise meaning and application of the parameters is explained below, in the descriptions of each of the three steps. To make Algorithm 2.2 more readable, they are not disclosed there.

Nodes of Cal5 trees may have different numbers of subnodes. The counts are automatically determined in the process of attribute discretization and interval merging.

Best Attribute Selection

Decisions, which attribute to select for node split generation, are made in Cal5 on the basis of a statistical approach or with entropy measure.

In the statistical method, each feature eligibility is estimated with the following quotient:

$$Q = \frac{A^2}{A^2 + D^2}, \tag{2.12}$$

Q criterion where D^2 is the mean value of squared variance of the classes with respect to their centroid vector, and A^2 is the mean value of squared distances between the centroids of the classes. An attribute with minimum Q value is selected as the best one.

The method based on entropy measure requires each attribute discretized, so in that case, the order between the steps of feature selection and discretization gets inversed. The discretization procedure, described below, is run for each feature and an index of weighted sum of entropies of the subsets is calculated. In fact, the index is just the information gain defined by Eqs. (2.4) and (2.5) and used in ID3. Naturally, the best attribute is the one with the largest information gain.

Discretization

The process of continuous attribute discretization starts with sorting the training data sample (assigned to the node) in the order of nondecreasing values of the attribute. Next, the intervals starting at $-\infty$ and ending between subsequent two adjacent values of the feature are analyzed. The analysis of an interval x is based on testing two hypotheses:

H1 – there exists a class $c \in \mathscr{C}$, such that $p(c|x) \geq s$,
H2 – for all classes $c \in \mathscr{C}$, $p(c|x) < s$.

The tests are done by calculation of the confidence interval $[d_1, d_2]$ for a specified level α, with the following formula derived from the Chebyshev's inequality with the assumption of Bernoulli distribution of each class:

$$d_{1/2} = \frac{2\alpha n_c + 1}{2\alpha n + 2} \mp \frac{1}{2\alpha n + 2}\sqrt{4\alpha n_c \left(1 - \frac{n_c}{n}\right) + 1}, \tag{2.13}$$

and check whether the whole interval lies on the adequate side of the threshold s. There are three possibilities:

1. Hypothesis H1 is true. Then, the interval is regarded as closed and it corresponds to a leaf assigned with the label of the class c which made H1 condition true. The analysis starts again for intervals beginning at the end of the interval just closed.
2. Hypothesis H2 is true. Then, the interval is also regarded as closed, but it corresponds to a node requiring further splits, because no class sufficiently dominates in the node.
3. Neither H1 nor H2 is true. Then, the interval is exceeded to include the next object from the ordered sample. If no more data objects are available, a leaf labeled with the dominating class is created.

Interval Merging

After discretization, adjacent intervals are merged if they both are leaves with the same class label. Adjacent intervals are also merged if no class dominates in them and they contain the same set of classes represented at least as frequently as in the case of the uniform distribution. The set of classes is determined by elimination of the classes for which $d_2 < \frac{1}{n_I}$, where n_I is the number of class labels occurring in the interval I.

Symbolic Features

Although Cal5 was designed to deal with data descriptions containing continuous features only, it is not very difficult to extend it to accept also symbolic attributes. When attribute selection is made with IG criterion, it can be applied naturally to symbolic features (discretization is just not necessary). Instead of division to intervals, groups of data objects sharing a value of discrete features can just be examined with hypothesis H1 and H2, to decide whether a node should become a leaf or needs further splits. It is also possible to apply the procedure of merging (designed for intervals) to the groups of data objects sharing a symbolic feature value. When rephrasing the algorithm, one needs to be careful about computational complexity of the resulting method, because in symbolic attributes there is no adjacency as in the case of intervals, and analyzing each pair of values may be costly. Sensible restrictions for the pairs of values considered for merging can be easily introduced in such a way, that computational complexity remains attractive. For example, relations between fractions of groups belonging to particular classes may be used as a substitute for intervals adjacency.

2.2.5 FACT, QUEST and CRUISE

A family of interesting DT algorithms has been created in the group of prof. Wei-Yin Loh. The family includes such algorithms as FACT (*Fast Algorithm for Classification Trees*, Loh and Vanichsetakul 1988), QUEST (*Quick, Unbiased, Efficient, Statistical Tree*, Loh and Shih 1997) and CRUISE (*Classification Rule with Unbiased Interaction Selection and Estimation*, Kim and Loh 2001, 2003). The methods are called "statistical trees" because they strongly base on statistical tools in tree construction and refinement. They have both univariate and multivariate forms, but the univariate algorithms are more often used so their descriptions are included here.

Algorithm 2.3 (DT induction by separate feature selection and split)

Prototype: FeatSelThenSplit(D)
Input: *Training dataset D, some configuration parameters.*
Output: *Node split.*
The algorithm:

```
$A \leftarrow BestAttribute(D)$
If $A = \perp$ return $\perp$
$s \leftarrow BestAttributeSplit(A, D)$
If $s = \perp$ return $\perp$
return $s$
```

The algorithms can be seen as classical top-down DT induction algorithms, that is, Algorithm 2.1 with a specific approach to best split selection (*BestSplit* procedure of the algorithm) divided into two parts: first the feature for split is selected and then particular split is searched for the selected feature, as in Algorithm 2.3. Thanks to independent feature selection and split, there is no need to perform exhaustive search for possible splits of a given node—only the splits of selected feature need to be analyzed. This may fasten the algorithm, if only the feature selection part is accurate. Otherwise the tree may become much larger and it may shatter the gains of the restrictions in split search space.

The authors paid much attention and undertook much effort to make their methods of feature selection unbiased. More detailed discussion of this subject is presented in Sect. 2.7, so here it is not further explored.

2.2.5.1 FACT

FACT (*Fast Algorithm for Classification Trees*, Loh and Vanichsetakul 1988) is designed to split datasets described by numeric features, but the authors provided a solution to convert symbolic attributes to continuous ones before the fundamental algorithm starts.

Conversion of Symbolic Features to Continuous Ones

To convert a discrete attribute X_D into a continuous X_C, FACT first creates a space of $n - 1$ binary features, where n is the number of possible values of the symbolic feature. Each dimension of the space is a binary indicator variable corresponding to one feature value (informs which objects originally had this value and which had not). Then, the largest *discriminant coordinate* (crimCoord, see appendix Sect. B.1, Gnanadesikan 1977) is found in this $n - 1$-dimensional space. It becomes a new continuous feature (X_C) replacing the original symbolic one (X_D). After the split is determined for the converted feature, it is easy to convert the conditions like $X_C > z$ to more informative form of $X_D \in A$.

Feature Selection

For univariate splits, FACT selects the feature by means of analysis of each attribute (discrete ones are analyzed after the conversion to continuous variables, described above) with the F statistic known from ANOVA (*analysis of variance*) methods, which is the ratio of between to within class variance (for more details see appendix section A.2.1 about *F-test* or (Tadeusiewicz et al. 1993; Brandt 1998)). The feature with the largest F ratio is selected, if only its F statistic exceeds a threshold F_0 (user-specified parameter of the method, with default value of 4). If $F < F_0$ each feature X is transformed to $Z = |X - \overline{X}|$ and F ratios for such transformed features are calculated. If the largest F ratio $F_Z \geq F_0$, than its feature Z is used for the split. Otherwise, the original feature X maximizing F ratio is used to generate a binary split with respect to $\overline{X}$.

FACT also offers an option to search for polar coordinate splits, more effective if there is an angular or radial symmetry in the data. It must be pointed out, however, that the feature constructed in this way makes the splits multivariate. If such option is selected anyway, the original data vectors are first converted into vectors of larger *principal components* (as presented below, in the description of the LDA-based split procedure). The resulting new features (linear combinations of the original ones) are subject to a similar analysis of F ratios, as in the case of original features, described above. As a result, either a linear combination feature is used for the split or a possibility of spherical symmetry detected. Since the symmetry may not be present in every variable, the original features are selected on the basis of Levene's homogeneity test of variances (see Levene 1960, appendix Sect. A.2.3), before they are transformed into polar coordinates.

Split Selection

FACT performs splits by means of *linear discriminant analysis* (LDA). The procedure is defined for multidimensional data descriptions. For univariate trees, the calculations get very simple. To avoid nonsingular covariance matrices a *principal component analysis* (PCA) is done at each node before the actual analysis. The components with eigenvalues smaller than β times the largest eigenvalue are rejected (β is a user-specified parameter). The remaining components take part in determination of the linear discriminant functions:

$$d_c(\mathbf{y}) = \mu_c^T \Sigma^{-1} \mathbf{y} - \frac{1}{2} \mu_c^T \Sigma^{-1} \mu_c + \ln p(c|N), \tag{2.14}$$

where $\mathbf{y}$ denotes a vector in the space of selected principal components μ_c is the sample mean vector of class c, Σ is the pooled estimate of the covariance matrix at node N and N is the node being split.

Nodes are split into as many subnodes as the number of classes represented within the node being split. Objects are delegated to the subnodes corresponding to the class minimizing the following formula respecting also a misclassification cost matrix $Cost$:

$$c_{win} = \arg\min_{c \in \mathscr{C}} \sum_{k \in \mathscr{C}} Cost(c|k) e^{d_k(\mathbf{y})}. \tag{2.15}$$

The technique is called *normal theory option*.

Stop Criterion to Improve Generalization

To prevent overfitting the training sample, FACT is equipped with a stop criterion, which is tested at each node. Further splits are not accepted if one of the following two conditions is met:

1. The node contains no more than one class represented by at least MINDAT objects (MINDAT is a user-specified parameter).
2. The split does not decrease predicted error rate. For a node N split into subnodes $N_1, \ldots, N_k$, denoting by c_X the class assigned to node X, the split is not accepted if

$$\sum_{c\in\mathscr{C}} Cost(c_N|c)p(c|N) \leq \sum_{i=1}^{k}\sum_{c\in\mathscr{C}} Cost(c_{N_i}|c)p(c|N_i). \tag{2.16}$$

FACT does not use any validation based pruning method similar to CART's cost-complexity minimization.

2.2.5.2 QUEST

Quick, Unbiased, Efficient, Statistical Tree (QUEST) algorithm (Loh and Shih 1997; Lim et al. 2000) was created as a significant improvement of FACT. The general idea and organization of the algorithm remained the same: the method realizes algorithm 2.3 separating feature selection from determination of the split, converts symbolic features to numeric ones in a similar way, and uses statistical tests to make some decisions. The main changes concern how the particular goals are obtained:

- split feature is selected on the basis of another approach to estimate feature importance, aimed at unbiased selection,
- the split is made with quadratic discrimination instead of linear,
- the resulting tree is binary, classes are grouped before the split,
- generalization is obtained with cost-complexity minimization, as in the case of CART.

Loh and Shih (1997) claim that the way they convert symbolic feature to continuous ones is also different in QUEST than in FACT, however they mention that FACT's method first converts the feature symbols into binary "dummy" vectors, and then converts them into real numbers with a method that can split the node into more than two subnodes, which is not acceptable in QUEST. They evidently refer to another version of FACT than the one of Loh and Vanichsetakul (1988), because as described above, the latter uses crimCoord transformation (see appendix section B.1) to convert symbols to numeric values, and the same is done in QUEST. Naturally, there is a difference between the two methods in the way they split the features after the conversion. As in FACT, after the split is determined for the continuous counterpart of a symbolic feature, it can be easily rephrased in the language of original symbols, so in the resulting tree, the continuous feature generated during the analysis is not at all visible.

Feature Selection

Estimation of both continuous and symbolic features with F ratio (for symbolic ones calculated for the derived continuous feature) is prone to more frequent selection of symbolic features than continuous ones, also when they are all independent from the target. The idea of QUEST to get rid of the bias (or at least to try to; see Sect. 2.7 for more discussion) is to compare p-values of independence tests most eligible for each type of features, instead of comparing the F ratios. Continuous attributes are still analyzed with F statistic, but discrete features are subject to the χ^2-test (no conversion to numeric values is performed at the stage of feature selection). The F and χ^2 values are not directly comparable, but comparing their p-values makes sense. The feature with the smallest p-value (the smallest probability of independence with the target variable) is selected as the best one. If none of the p-values exceeds a user-defined threshold parameter, Levene's F-test for unequal variances is computed for each ordered variable. The tests thresholds are Bonferroni corrected (see lines 5 and 5d of algorithm 2.4 and appendix section A.1.4). If the best result exceeds the threshold, the corresponding feature is selected, otherwise the algorithm returns the feature with the smallest p-value calculated in the first stage (with F-test or)χ^2-test.The method is written formally as Algorithm 2.4.

Algorithm 2.4 (QUEST split feature selection)

Prototype: ***QUESTFeatSel(D,*** α***)***

Input: *Training dataset D described by discrete features* $X_1, \ldots, X_d$ *and continuous features* $X_{d+1}, \ldots, X_f$*, confidence threshold* α*.*

Output: *Feature index.*

The algorithm:

```
1. If $d > 0$ (there are continuous features)
    a. for $i = 1, \ldots, d$
        $F_i \leftarrow$ the ANOVA $F$ statistic for feature $X_i$
    b. $best_1 \leftarrow \arg\max_{i=1,\ldots,d} F_i$
    c. $\alpha_1 \leftarrow$ p-value of the adequate $F$ distribution for feature $best_1$
2. If $f > d$ (there are discrete features)
    a. for $i = d + 1, \ldots, f$
        $p_i \leftarrow$ p-value of the $\chi^2$-test of independence between feature $X_i$ and class labels
    b. $best_2 \leftarrow \arg\min_{i=d+1,\ldots,f} p_i$
    c. $\alpha_2 \leftarrow p_{best_2}$
3. $\alpha_{12} \leftarrow \min(\alpha_1, \alpha_2)$
4. If $\alpha_{12} = \alpha_1$ then $best_{12} \leftarrow best_1$ else $best_{12} \leftarrow best_2$
5. If $\alpha_{12} \geq \frac{\alpha}{f}$ (no feature is good enough)
    a. for each $i = 1, \ldots, d$
        $F_i' \leftarrow$ the ANOVA $F$ statistic for feature $Z_i = |X_i - \overline{X_i}|$
    b. $best_3 \leftarrow \arg\max_{i=1,\ldots,d} F_i'$
    c. $\alpha_3 \leftarrow$ p-value of the adequate $F$ distribution for feature $best_3$
    d. If $\alpha_3 < \frac{\alpha}{f+d}$ then return $best_3$
6. return $best_{12}$
```

Split Selection

QUEST finds the best split points with *quadratic discriminant analysis* (QDA). Because QUEST is assumed to be a binary tree, the splits are done between two classes. If the problem at hand concerns classification to more than two classes, they are first grouped into two superclasses, with the clustering*two-means clustering* method (Hartigan and Wong 1979) applied to the set of mean vectors calculated for all the classes. The two-means algorithm is initialized with two most distant class means as the cluster centers. If all the class means are identical, then the most populous class composes superclass A and the rest—superclass B.

In the case of continuous features, a procedure of QDA is directly performed, and in the case of symbolic ones the crimCoord based procedure is run to obtain a continuous substitute for the feature, which is then analyzed with QDA.

The QDA estimates the two classes (A, B) distributions with normal densities and determines the split point as the point of intersection of the two Gaussian curves, being a root of the equation

$$P(A|N)\frac{1}{\sqrt{2\pi}s_A}e^{-\frac{(x-\overline{x}_A)^2}{2s_A}} = P(B|N)\frac{1}{\sqrt{2\pi}s_B}e^{-\frac{(x-\overline{x}_B)^2}{2s_B}}, \tag{2.17}$$

where N is the node being split, $\overline{x}_A, \overline{x}_B$ are the means of class A and B respectively and s_A, s_B are standard deviations observed within the classes. The normal densities parameters (means and standard deviations) are calculated from the samples. Equation (2.17) after simple transformations gets the form of a typical quadratic equation:

$$ax^2 + bx + c = 0, \tag{2.18}$$

where

$$\begin{aligned} a &= s_A^2 - s_B^2, \quad b = 2(\overline{x}_A s_B^2 - \overline{x}_B s_A^2), \\ c &= (\overline{x}_B s_A)^2 - (\overline{x}_A s_B)^2 + 2s_A^2 s_B^2 \log \frac{n_A s_B}{n_B s_A}. \end{aligned} \tag{2.19}$$

The split point is one of the two roots that is closer to $\overline{x}_A$, provided this yields two nonempty subsets.

2.2.5.3 CRUISE

Classification Rule with Unbiased Interaction Selection and Estimation (CRUISE, Kim and Loh 2001, 2003) is a descendant of FACT and QUEST. It is the next significant step towards unbiased feature selection. As its predecessors, CRUISE also fits the general strategy of Algorithm 2.3, but differs in many detailed solutions:

- generates multi-split trees,
- introduces new method for split feature selection, named 2D because of analysis of pairs of features (more precisely some contingency tables), but still supports the method of QUEST (here named 1D) based on comparing p-values of proper independence tests for each type of features (ANOVA F-test for continuous ones and χ^2-test for categorical),

- uses Box-Cox power transformation to adjust class distributions in order to improve the accuracy of LDA.

Similarly to FACT and QUEST, categorical features are transformed with CrimCoord at the start of the analysis and Bonferroni corrections (see appendix section A.1.4) are used when performing multiple statistical tests. Missing values in the data can be ignored or imputed. As the main advantages of CRUISE its authors mention its sensitivity to local interactions between variables (thanks to)2D analysis, which results in more intelligent splits and shorter trees, and its speed obtained in parallel with not statistically significant difference in mean misclassification rates, in comparison to the best methods.

CRUISE can be configured to generate linear combination splits as well as univariate splits.

Feature Selection

Two different methods of feature selection are available in CRUISE. They are named "1D" and "2D" respectively. 1D is exactly the same method as the one used in QUEST. The novelty is the second method based on two-dimensional contingency-tables analysis. This method performs statistical tests for five different types of sources (two marginal tests and three interaction tests). For each case, the space is split into a number of regions and a contingency table with classes as rows and the regions as columns is created. The possible sources and the ways of the contingency tables construction are the following:

- for single numeric variable X, four regions correspond to the sample quartiles of X,
- for single categorical variable, the regions correspond to the values of the variable,
- for a pair of numeric variables, the regions correspond to four quadrants resulting from splits at the sample medians,
- for a pair of categorical variables, the regions correspond to pairs of possible values of the variables,
- for a pair of one numeric and one categorical variable, the regions correspond to combinations of two parts of the numeric attribute (split at the median) and all possible categories of the categorical feature.

The procedure of building the contingency table for each context is noted in Algorithm 2.6 as *ContTable()* and its arguments clearly show which of the five versions is called. Testing contingency tables for pairs of variables is aimed at detecting interactions between features, but the possibilities are significantly limited, because splitting numeric features arbitrarily at medians, can correspond to proper decision borders only accidentally.

Each contingency table is analyzed with Algorithm 2.5 to get a corresponding z-value. Maximum of the 5 z-values (with a bootstrap bias correction factor) points the selected feature according to the rules presented in the algorithm 2.6.

The *BootstrapCorrection()* function used in the algorithm, finds a factor $f \in R$, that brings proper balance between discrete and continuous features (eliminates the

Algorithm 2.5 (z-statistic of a contingency table)

```
Prototype: ZStatistic(T)
Input: Contingency table T with n rows and m colums.
Output: z-statistic.
The algorithm:
  1. $\chi^2 \leftarrow$ the Pearson $\chi^2$ statistic with $\nu = (n-1)(m-1)$ degrees of freedom for $T$
  2. $W \leftarrow \chi^2 - \nu + 1$
  3. if $z > 1$ then (Peizer-Pratt transformation)
       $z \leftarrow \frac{1}{|W|}(W - \frac{1}{3})\sqrt{(\nu - 1)\log\frac{\nu-1}{\chi^2} + W}$
     else
       $z \leftarrow \sqrt{\chi^2}$
  4. return $z$
```

Algorithm 2.6 (CRUISE 2D feature selection)

```
Prototype: CRUISE2DFeatSel(D, $\alpha$)
Input: Training dataset $D$ described by discrete features $\mathbf{X}_1, \ldots, \mathbf{X}_d$ and continuous features
       $\mathbf{X}_{d+1}, \ldots, \mathbf{X}_f$ and targets $\mathbf{Y}$.
Output: z-statistic.
The algorithm:
  1. if $d = 0$ then $z_d \leftarrow -\infty$
     else $z_d \leftarrow \max_{i=1,\ldots,d}$ ZStatistic(ContTable($\mathbf{Y}; \mathbf{X}_i$))
  2. if $f = d$ then $z_c \leftarrow -\infty$
     else $z_c \leftarrow \max_{i=d+1,\ldots,f}$ ZStatistic(ContTable($\mathbf{Y}; \mathbf{X}_i$))
  3. if $d = 0$ then $z_{dd} \leftarrow -\infty$
     else $z_{dd} \leftarrow \max_{i,j=1,\ldots,d}$ ZStatistic(ContTable($\mathbf{Y}; \mathbf{X}_i, \mathbf{X}_j$))
  4. if $f = d$ then $z_{cc} \leftarrow -\infty$
     else $z_{cc} \leftarrow \max_{i,j=d+1,\ldots,f}$ ZStatistic(ContTable($\mathbf{Y}; \mathbf{X}_i, \mathbf{X}_j$))
  5. if $d = 0$ and $f = d$ then $z_{dc} \leftarrow -\infty$
     else $z_{dc} \leftarrow \max_{i=1,\ldots,d;\, j=d+1,\ldots,f}$ ZStatistic(ContTable($\mathbf{Y}; \mathbf{X}_i, \mathbf{X}_j$))
  6. $f \leftarrow$ BootstrapCorrection($D$)
  7. $z \leftarrow \max\{z_d, f z_c, z_{dd}, f z_{cc}, z_{dc}\}$
  8. if $z = z_d$ or $z = f z_c$ then return the feature maximizing $z$
  9. if $z = z_{dd}$ or $z = f z_{cc}$ then return the feature in the interacting pair with larger marginal
     $z$
 10. return the categorical feature in the interacting pair
```

bias). Bootstrap data samples are generated (the data input is copied and the targets are bootstrapped to make them independent from the input) repeatedly and the 5 z-values are calculated for each bootstrap sample as in Algorithm 2.6. A win is scored for continuous features if $f \max\{z_c, z_{cc}\} \geq \max\{z_d, z_{dd}, z_{dc}\}$ and for discrete features otherwise. Several values of f are tested and for each, the proportion between the wins of continuous and discrete features is calculated. Eventually, if necessary, linear interpolation is used to find the final result: such value of f that its proportion of wins is equal to the proportion of the counts of continuous and discrete features describing the data.

Split Selection

When a continuous feature is selected for a split, Box-Cox power transformation (see appendix section B.4) is run to make the feature more adequate for linear discriminant analysis (LDA) which decides about the split in the same way as it does in FACT.

If the feature selection stage is won by a categorical variable $\mathbf{X} \in \mathscr{X}^n$, where $\mathscr{X} = \{a_1, \ldots, a_r\}$, then it is converted into r binary indicator variables, so that each value a_i is converted into a unit vector $e_{\mathscr{X}}(a_i) = e_i$ with 1 at position i and 0 at all the others. Such vectors are then projected onto the largest discriminant coordinate (CrimCoord) and the new dimension is treated in the same way as numeric attributes: it is passed to the Box-Cox transformation and split with LDA. The final result is translated to the language of the original, discrete feature, to make it comprehensible.

This is the procedure performed by the 2D method. In 1D, there is a difference that the Box-Cox transformation is not run if the feature was selected by Levene's test. Then, instead, LDA is applied to the absolute deviations from the sample mean at the node.

Because LDA happens to generate splits with no data within, such intervals are divided into halves and merged with their neighbors.

2.2.6 CTree

The pursuit of unbiased feature selection in DT construction has gained many different solutions. One of the most interesting results is the approach of Hothorn et al. (2004, 2006a, 2008) and Zeileis et al. (2008) to a conditional inference framework, capable of unbiased recursive partitioning. The authors found the work of Strasser and Weber (1999) on permutation statistics very useful in DT induction.

The main idea behind the framework of *CTree* is the same as in the case of the FACT family, and as depicted by algorithm 2.3, where feature selection and split finding are separate processes. Here, for feature eligibility estimation, and possibly for best split determination, non-parametric permutation tests are used (in place of the F-test and χ^2-test of the FACT line).

Feature Selection and Stop Criterion

The same permutation tests used for feature selection are helpful in deciding when to stop further splits of a node. The null hypothesis of interest is that the unconditional distribution of the target variable $\mathbf{Y}$ and its conditional distributions with respect to each covariate $\mathbf{Y}|\mathbf{X}_i$ are the same. If we are not able to reject such hypothesis at a tree node, the node should be closed into a leaf with no further splits. Otherwise, the input variable $\mathbf{X}_{i^*}$ providing the least independence of the two distributions (the least p-value) is selected as the best feature for the split.

Given a data sample D of n objects described by m features $\mathbf{X}_1 \in \mathscr{X}_1^n, \ldots, \mathbf{X}_m \in \mathscr{X}_m^n$ and the target variable $\mathbf{Y} = (Y_1, \ldots, Y_n) \in \mathscr{Y}^n$, and a case weight vector $\mathbf{w} \in R^n$, Hothorn et al. (2006b) proposed to measure the association between $\mathbf{Y}$ and $\mathbf{X}_j$, $j = 1, \ldots, m$, by linear statistics of the form:

$$\mathbf{T}_j(D, \mathbf{w}) = vec\left(\sum_{i=1}^{n} w_i g_j(X_{ji}) h(Y_i, \mathbf{Y})^T\right) \in R^{p_j q}, \tag{2.20}$$

where:

- $g_j : \mathscr{X}_j \to R^{p_j}$ is a transformation of feature $\mathbf{X}_j$ (for example, to convert symbolic features to more reasonable form like binary vectors),
- $h : \mathscr{Y} \times \mathscr{Y}^n \to R^q$ is the *influence function* dependent on the responses $\mathbf{Y}$ in a permutation symmetric way,
- vec operator converts the $p_j \times q$ matrix it gets as the argument to a $p_j q$-dimensional vector by column-wise concatenation.

This is a general definition that can be used in both classification and regression tasks with miscellaneous definitions of the feature spaces $\mathscr{X}_j$ and $\mathscr{Y}$ and proper substitutions for functions g_j and h. For example, for the sake of univariate classification trees:

- the features are either continuous (real numbers) or symbolic,
- the target variable space is $\mathscr{Y} = \mathscr{C} = \{c_1, \ldots, c_k\}$,
- the function h can be defined as

$$h(y, \mathbf{Y}) = e_{\mathscr{C}}(y), \tag{2.21}$$

  that is, a k-dimensional unit vector with 1 at the position i such that $y = c_i$ (the dimensions are binary indicator variables),
- for $j = 1, \ldots, m$, the function g_j can be defined as:

$$g_j(x) = \begin{cases} x & \text{if} \mathscr{X}_j = R \\ e_{\mathscr{X}_j}(x) & \text{if} \mathscr{X}_j = \{a_1, \ldots, a_r\}, \end{cases} \tag{2.22}$$

  where $e_{\mathscr{X}_j}(x)$ is an r-dimensional unit vector with 1 at index i such that $x = a_i$.

Under the null hypothesis H_0^j, that the distributions of $\mathbf{Y}$ and $\mathbf{Y}|\mathbf{X}_j$ are identical, the distribution of $\mathbf{T}_j(D, \mathbf{w})$ can be analyzed by means of *permutation tests*. In place of the dependency of $\mathbf{T}_j(D, \mathbf{w})$ on usually unknown joint distribution of $\mathbf{Y}$ and $\mathbf{X}_j$, one can fix the covariates and condition on all possible permutations of the responses. Following Strasser and Weber (1999), the conditional expectation $\mu_j \in R^{p_j q}$ and covariance matrix $\Sigma_j \in R^{p_j q \times p_j q}$ of $\mathbf{T}(D, \mathbf{w})$, given all permutations $\sigma \in S(D, \mathbf{w})$ are:

$$\mu_j = E(\mathbf{T}_j(D,\mathbf{w})|S(D,\mathbf{w})) = vec\left(\left(\sum_{i=1}^{n} w_i g_j(X_{ji})\right) E(h|S(D,\mathbf{w}))^T\right), \tag{2.23}$$

$$\begin{aligned}\Sigma_j &= V(\mathbf{T}_j(D,\mathbf{w})|S(D,\mathbf{w})) \\ &= \frac{w_\Sigma}{w_\Sigma - 1} V(h|S(D,\mathbf{w})) \otimes \left(\sum_{i=1}^{n} w_i g_j(X_{ji}) \otimes w_i g_j(X_{ji})^T\right) \\ &\quad - \frac{w_\Sigma}{w_\Sigma - 1} V(h|S(D,\mathbf{w})) \otimes \left(\sum_{i=1}^{n} w_i g_j(X_{ji})\right) \otimes \left(\sum_{i=1}^{n} w_i g_j(X_{ji})^T\right)\end{aligned} \tag{2.24}$$

where $w_\Sigma = \sum_{i=1}^{n} w_i$ and $\otimes$ is the Kronecker product and the conditional expectation and covariance matrix of the influence function are:

$$E(h|S(D,\mathbf{w})) = \frac{1}{w_\Sigma} \sum_{i=1}^{n} w_i h(Y_i, \mathbf{Y}) \in R^q, \tag{2.25}$$

$$\begin{aligned}V(h|S(D,\mathbf{w})) = \frac{1}{w_\Sigma} \sum_{i=1}^{n} & w_i (h(Y_i,\mathbf{Y}) - E(h|S(D,\mathbf{w}))) \\ & (h(Y_i,\mathbf{Y}) - E(h|S(D,\mathbf{w})))^T.\end{aligned} \tag{2.26}$$

On the basis of the pq-dimensional statistic $\mathbf{T}$, the final test statistic c can be defined in an arbitrary way. The most natural solution for univariate statistic, suggested by Hothorn et al. (2006b) is:

$$c_{\max}(\mathbf{t}, \mu, \Sigma) = \max_{i=1,\ldots,pq} \left| \frac{(\mathbf{t}-\mu)_i}{\sqrt{\Sigma_{ii}}} \right|. \tag{2.27}$$

Another possibility is a more computationally expensive quadratic form:

$$c_{quad}(\mathbf{t}, \mu, \Sigma) = (\mathbf{t}-\mu)\Sigma^{+}(\mathbf{t}-\mu)^T. \tag{2.28}$$

In any case, one must be aware that the test statistics c may not be directly comparable. In such circumstances, p-values should be calculated, because they allow for unbiased feature selection. Naturally, the way of calculating p-values is closely bound up with the definition of the c statistic. From theorems proved by Strasser and Weber (1999) it can be derived that asymptotic conditional distribution of c_{max} is normal. Quadratic c_{quad} follows asymptotic χ^2 distribution with degrees of freedom given by the rank of Σ.

To get precise stop criterion, the overall null hypothesis being the conjunction of all hypotheses H_0^j needs to be verified. One can construct and analyze an aggregate statistic for this purpose, but in practice it is preferred to use simple techniques like Bonferroni correction for multiple testing. A significance level α must be provided to control the pre-pruning. Hothorn et al. (2006b) suggest the default value of $\alpha = 0.05$, but the meta-parameter can also be optimized in a validation-based procedure similar to cost-complexity optimization cost-complexity minimization of CART.

Split Selection

When the feature of split is selected, different ways of finding the best split of the feature can be applied including those of CART, FACT, QUEST and many others. The splits can be binary or multi-way, accordingly. After such suggestion, Hothorn et al. (2006b) proposed another application of the permutation test framework to determine optimal binary splits. According to their approach, given a subset A of the sample space $\mathscr{X}_j$, the linear statistic $\mathbf{T}_j^A$ is defined as

$$\mathbf{T}_j^A = vec\left(\sum_{i=1}^{n} w_i 1_A(X_{ji}) h(Y_i, \mathbf{Y})^T\right) \in R^q, \tag{2.29}$$

where 1_A is the indicator function of set A. Given the conditional expectation μ_j^A and its covariance Σ_j^A calculated with Eqs. (2.23) and (2.24), the optimum split is determined by the set $A^\star$, such that

$$A^\star = \arg \max_{A \subset \mathscr{X}_j} c(\mathbf{t}_j^A, \mu_j^A, \Sigma_j^A). \tag{2.30}$$

Although the optimization extends over all subsets of $\mathscr{X}_j$, the number of binary splits is usually significantly limited. In the case of numeric features, only splits into two disjoint and complementary intervals defined by points between values represented in the data are taken into account. Symbolic features with large number of possible values may also need some restrictions in subset analysis, because of computational complexity.

2.2.7 SSV

Separability of Split Value (SSV, Grąbczewski and Duch 1999, 2000) criterion is defined as a split quality measure, but is not based on the purity gain rule (2.4). It reflects the idea that splitting pairs of vectors belonging to different classes is advantageous, while splitting pairs of vectors of the same class should be avoided if possible. Originally, it has got two forms:

$$\mathrm{SSV}(s, D) \overset{def}{=} 2 \cdot \mathrm{SSV}_1(s, D) - \mathrm{SSV}_2(s, D), \tag{2.31}$$

$$\mathrm{SSV}_{lex}(s, D) \overset{def}{=} \Big(\mathrm{SSV}_1(s, D), -\mathrm{SSV}_3(s, D)\Big), \tag{2.32}$$

where:

$$\mathrm{SSV}_1(s,D) \overset{def}{=} \sum_{i=1}^{n_s} \sum_{j=i+1}^{n_s} \sum_{c\in\mathscr{C}} |D_{s_i,c}| \cdot |D_{s_j} \setminus D_{s_j,c}|, \tag{2.33}$$

$$\mathrm{SSV}_2(s,D) \overset{def}{=} \sum_{c\in\mathscr{C}} (|D_c| - \max_{i=1,\dots,n_s} |D_{s_i,c}|), \tag{2.34}$$

$$\mathrm{SSV}_3(s,D) \overset{def}{=} \sum_{i=1}^{n_s} \sum_{j=i+1}^{n_s} \sum_{c\in\mathscr{C}} |D_{s_i,c}| \cdot |D_{s_j,c}|. \tag{2.35}$$

The SSV_1 part counts the pairs of separated objects belonging to different classes—it can be called a reward part. SSV_2 and SSV_3 define, in two different ways, some penalties for splitting objects representing the same class. The $\mathrm{SSV}_{\mathrm{lex}}$ version provides pairs of values, which are compared in lexicographic order, so the second value is considered only in the case of equal first elements.

It is not easy to keep proper balance between the reward part and the penalty part of the criteria like (2.31) or (2.32), because repairing some cases may easily spoil the functionality in other cases. Some analyses of special cases, brought an idea to weight the pairs of separated objects when counting the separability index. Weighting can be seen as a heuristic reflecting the fact that separating pairs of objects is more advantageous, when the objects belong to the majority classes within their sides of the split, and less valuable if the objects are still misclassified after the split. Therefore a parameter weight α was introduced (Grąbczewski 2011) as a factor to diminish the contribution of the minority objects in separated pairs. The result is the following definition:

$$\mathrm{SSV}_\alpha(s,D) \overset{def}{=} \sum_{i=1}^{n_s} \sum_{j=i+1}^{n_s} \sum_{\substack{a\in\mathscr{C}\\ b\in\mathscr{C}\\ a\neq b}} W_\alpha(D_{s_i},a) \cdot |D_{s_i,a}| \cdot W_\alpha(D_{s_j},b) \cdot |D_{s_j,b}|, \tag{2.36}$$

where

$$W_\alpha(D,c) = \begin{cases} 1 & \text{if } c \text{ is the majority class within } D, \\ \alpha & \text{otherwise.} \end{cases} \tag{2.37}$$

Such definition introduces three levels of contribution of the separated pairs (1, α and α^2), dependent on whether the objects represent the majorities or not. If more than one class is represented in a sample with maximum count, one of them is arbitrarily selected as the majority class (in practice, the one with the smallest index).

Search Methods

The basic approach to searching for SSVtrees is the classical top-down induction method presented as Algorithm 2.1 with an almost exhaustive search for each node split. The term “almost exhaustive” means that all possible splits are examined if only it is acceptable from the point of view of computational costs and does not introduce evident bias in feature selection (favoring symbolic features with many possible values).

In the case of continuous features, all sensible split points are examined and the one maximizing SSVselected. Here, “sensible” means the ones with nonzero probability that they can bring the best result. It is obvious that only the points between the feature values occurring in the node data are worth any interest. It can be easily proved that the points between objects belonging to the same class can be omitted, because some other points are certainly better. So the analysis procedure of a numeric feature starts with sorting all sample objects by the values of the feature and exploring the sorted values one by one and calculating SSVfor the “sensible” split points.

Symbolic features are split not with points but with subsets of symbols, hence the general term “value” in the name SSV. Exhaustive search would test all possible subsets as the split generators. Such algorithm complexity is exponential, so it can be dangerous to run it for larger counts of feature symbols. It would also give symbolic features significantly greater probability of selection, in comparison to continuous ones, when they are similarly informative. To avoid such bias and the danger of combinatorial explosion, SSVuses a *subset pair enumerator* with a possibility to limit the number of enumerated splits. The enumerator provides subsequent pairs of complementary subsets, by generating the first one and setting the other as the complement to the whole set of symbols. Preference is given to smaller subsets (against their complements), so at first, the singletons are handled then pairs and so on. The limit can be set on the size of the first subset. SSVsets the limit to make the number of tested splits as close to the one of continuous features (and the number of objects in the node) as possible.

Apart from the most popular method of tree construction, based on hill climbing, the SSVapproach has been successfully tested with *beam search* and *lookahead search* (see Sect. 2.5). Both solutions are more computationally expensive than hill climbing, but often can find smaller trees. Different explorations of the area of search methods (Quinlan and Cameron-Jones 1995; Segal 1996; Janssen and Fürnkranz 2009) have shown that these approaches can be successful, but also can lead to so called “oversearching”, so one must use these methods with caution.

2.2.8 ROC-Based Trees

Receiver Operator Characteristic (ROC, Green and Swets 1966) is an idea that has found especially much interest in domains close to medicine, where it is very important to differentiate between false positive and false negative answers (statistical *Type I error*type Itype I and *type II error*). When the null hypothesis is the diagnosis of “healthy”, erroneous rejecting the hypothesis means a false alarm (False positive answer), while failed rejection of the hypothesis, when it is not true (type II error) means ignoring the illness and no medical treatment, when it should be undertaken, which can be much more serious. One of the solution of dealing with such differences in errors importance is considering misclassification costs (available in many classification learning machines). Clinicians are usually interested in the information about how many errors of each kind are made by the decision support system, they use. Therefore, they find ROC curves very useful, because they are plots visualizing numbers of erroneous answers of both kinds, given by classifiers. In binary classification, the decisions can be divided into four groups:

Predicted class	Original class	
	Positive	Negative
Positive	True positive (TP)	False positive (FP)
Negative	False negative (FN)	True negative (TN)

On the basis of the four groups we can define many important performance indices, for example:

$$\begin{aligned} accuracy &\overset{def}{=} \frac{TP+TN}{TP+FP+FN+TN}, & sensitivity &\overset{def}{=} \frac{TP}{TP+FN}, \\ error &\overset{def}{=} \frac{FP+FN}{TP+FP+FN+TN}, & specificity &\overset{def}{=} \frac{TN}{FP+TN}. \end{aligned} \tag{2.38}$$

All the values are real in the range [0, 1]. Intuitions about the terms of accuracy and error are common and obvious. *Sensitivity* (*Se*, also called *true positive rate*) shows which part of the "positive" class is correctly detected (in medicine: how big part of patients with the disease is correctly diagnosed). *Specificity* (*Sp*, also called *true negative rate*) tells, how accurately the negative cases are recognized. In medicine, even more important is the value of "1-specificity" (*false positive rate*, or *fall-out*), which represents the amount of negative cases classified as positive.

It is easy to increase the sensitivity of a predictor paying the price of lower specificity, and inversely. In utter cases, classification of all objects as positive results in 100 % sensitivity but 0 % specificity, while constant "negative" answers are 100 % specific but 0 % sensitive. The art of learning is to maximize both sensitivity and specificity.

ROC curve is a line plot depicting transition from one extremity to the other through the most valuable solutions. The axes of the plot are: *1-specificity*specificity and *sensitivity*. Assuming two-element set of classes $\mathscr{C} = \{positive, negative\}$, each classifier is a function $\phi : O \rightarrow \{positive, negative\}$ and can be visualized as a single point in the ROC plot.

For a probabilistic classifier $\phi : O \rightarrow R^2$, we can easily generate a series of crisp classifiers by shifting the decision threshold θ:

$$\phi^{(\theta)} : O \rightarrow \mathscr{C}, \quad \phi^{(\theta)}(o) = \begin{cases} positive & \text{if } \phi(o)_1 \geq \theta, \\ negative & \text{otherwise.} \end{cases} \tag{2.39}$$

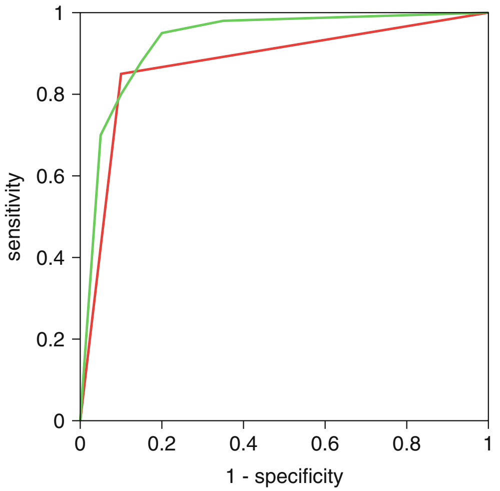

Fig. 2.1 Two example ROC curves: one for crisp and one for probabilistic classifier

Each crisp classifier can be easily converted into a trivial probabilistic classifier (with probabilities 0 or 1 only). Such models can be visualized by an ROC curve based on three points: (0,0), (Se, 1-Sp), (1,1). Each ROC curve starts at point (0,0) and ends at (1,1).

Two example ROC curves are presented in Fig. 2.1. One corresponds to a crisp classifier and one to a probabilistic classifier with several levels of predicted probabilities. The curves are nondecreasing (here, also concave, though not strictly, as they are piecewise linear), because in such families of classifiers, increasing sensitivity is closely bound up with nonincreasing specificity.

In many applications the *area under the ROC curve* (AUC) has been found very attractive index of classifier (family) quality. The larger the AUC, the better the classifier.

AUC Split Criterion

Measuring the area under ROC curve has found applications also in DT construction. Ferri et al. (2002) started their road to *AUCsplit* criterion with an analysis of decision tree models in a similar manner as the transition from crisp to probabilistic classifier described above. They discuss possibilities of different labeling of DT leaves and prove formally an intuitive property that the number of optimal labelings is linearly, not exponentially dependent on the number of leaves. They order the leaves by *local positive accuracy* defined as $\frac{p_N}{p_N+n_N}$, where p_N and n_N are the counts of objects in the node N with labels "positive" and "negative" respectively, and show that optimal labelings are those that give label "positive" to a number of beginning nodes in the sequence and label "negative" to the rest of the nodes. In this way, they obtain a collection of points $P_0, \ldots, P_k$, such that

$$\underset{i=1,\ldots,k}{\forall} P_i = P_{i-1} + \left(\frac{n_i}{n}, \frac{p_i}{p}\right) = \left(\frac{\sum_{j=1}^{i} n_j}{n}, \frac{\sum_{j=1}^{i} p_j}{p}\right), \tag{2.40}$$

where k is the number of leaves in the tree, p_i and n_i are the numbers of objects with labels "positive" and "negative" respectively in $i'th$ DT leaf of the ordered collection, while p and n are the respective sums for all leaves. The points define the ROC curve, which is a piecewise linear function.

The area under such curve can be easily calculated as the sum of the areas of subsequent trapezia.

$$AUC(P_0, \ldots, P_k) = \sum_{i=1}^{k} \frac{y_i + y_{i-1}}{2}(x_i - x_{i-1}), \tag{2.41}$$

where $P_i = (x_i, y_i)$.

This idea is a foundation of the *AUCsplit* criterion (Ferri et al. 2002). Each split s of a node N yields a number of subnodes: $N_1^s, \ldots, N_{n_s}^s$. When the subnodes are sorted by local positive accuracy, they determine ROC points $P_0^s, \ldots, P_{n_s}^s$. The $AUCsplit$ criterion is defined as

$$AUCsplit(s) = AUC(P_0^s, \ldots, P_{n_s}^s). \tag{2.42}$$

It can be used to estimate quality of candidate splits and select the best split of given tree node in the classical top-down DT induction procedure (Algorithm 2.1).

In the case of a crisp classifier, the ROC is determined by three points $(0, 0)$, $(1 - Sp, Se)$, $(1, 1)$, where Sp and Se are the classifier's specificity and sensitivity. The AUC of such ROC is equal to $\frac{1}{2}(Se + Sp)$ which is the same as *balanced accuracy* (in two-class problems). It gives some specific view of the $AUCsplit$ optimization.

Ferri et al. (2002) have tested DTs based on their criterion in combination with the Pessimistic Error Pruning, described in more detail in Sect. 2.4.2.1, but any other pruning method can also be used to increase generalization abilities of the trees.

One of the most serious drawbacks of the *AUCsplit* criterion is that it can be used only for two-class problems. Although it is not difficult to generalize the ideas to arbitrary number of classes, the authors have not proposed such generalizations. Instead they combined the method with the 1-vs-1 strategy (Hand and Till 2001; Ferri et al. 2003) and proposed some improvements to provide more attractive probability estimates from the trees. Doetsch et al. (2009) used the criterion with maximization of the criterion over all class pairs in their Logistic Model Trees.

2.3 Multivariate Decision Trees

Splitting decision tree nodes on the basis of univariate functions is very attractive because of models comprehensibility. A price to pay for the comprehensibility is not too flexible shape of decision borders. Univariate conditions correspond to separation hyperplanes perpendicular to the axes of the selected features. When the splits are allowed to base on hyperplanes without restrictions, the node conditions may contain linear combinations in place of single features. Resulting trees are usually simpler in the sense of the number of nodes or leaves, but it is important to realize that the nodes are more complex so the overall tree complexity is not necessarily lower. At the same time, it gets much more difficult to interpret the resulting classification functions.

Many DT induction algorithms facilitate building both kinds of DTs by proper parameter settings. For example, some of the algorithms described above as the univariate methods (CART, FACT, QUEST and CRUISE) can also determine linear combinations of original features as new variables to be split. Many others have been especially designed to perform multivariate splits. In subsections below, a subjective selection of algorithms has been presented in more detail, however it must be pointed out that many other interesting approaches can also be found in the literature. It should certainly be recommended to also read the original publications about such methods like SADT (Heath et al. 1993), SPRINT (Shafer et al. 1996), CLOUDS (Alsabti et al. 1998), CMP (Wang and Zaniolo 2000), SODI (Lee and Olafsson 2006), Cline (Amasyali and Ersoy 2008) and many others.

2.3.1 LMDT

One of the first, very popular approaches to building DTs with splits based on linear combinations of features is the LMDT algorithm (Utgoff and Brodley 1991; Brodley and Utgoff 1992a,b). Similarly to the univariate trees described above, the main engine of LMDT is Algorithm 2.1. The main difference is inside the *BestSplit* procedure used in the approaches. LMDT does not select a feature for the split or analyze the dataset feature by feature to find the best split, but builds a *linear machine* for each node to split it. Each linear machine is a set of k linear discriminant functions combined in a single machine to classify data objects to one of the k classes of the problem being solved ($\mathscr{C} = \{c_1, \ldots, c_k\}$). Let $O \subseteq R^m$ be the domain of the classification task. The discriminant functions of the machine are

$$g_i : O \times \{1\} \rightarrow R, \quad g_i(\mathbf{y}) = \mathbf{w}_i^T \mathbf{y} \quad i = 1, \ldots, k. \tag{2.43}$$

The input vectors are extended by one dimension with constant value of 1 to facilitate versatility of hyperplanes definition. The linear machine classification function is

$$\phi_{\mathbf{g}} : O \rightarrow \mathscr{C}, \qquad \phi_{\mathbf{g}}(\mathbf{x}) = c_i \Leftrightarrow \underset{j \neq i}{\forall}\, g_j(\mathbf{x}, 1) < g_i(\mathbf{x}, 1). \tag{2.44}$$

In theory, when no unique maximum value of $g_i(\mathbf{x}, 1)$ exists, the value of the linear machine is undefined. Practical implementations usually select arbitrarily one of the joint winning classes if there is a draw, because it is more advantageous in usual classification tests to select one of the classes and have some chance to guess the result, than to give up because of equal probabilities of two classes.

The training process of an LMDT linear machine, tests randomly selected training vectors, and if the class assigned to the vector by the machine is incorrect (say c_j instead of c_i), then the weight vectors $\mathbf{w}_i$ and $\mathbf{w}_j$ are adjusted appropriately, to correct the classification of the vector. The detailed procedure is presented as Algorithm 2.7.

Algorithm 2.7 (Thermal linear machine training process)

```
Prototype: LMTraining(D,a,b)
Input: Training dataset $D \subseteq O \times \mathscr{C}$ ($O \subseteq R^m$), thermal parameter adjustment values a and b
        (default $a = 0.995$, $b = 0.0005$).
Output: Linear machine $\phi_{\mathbf{g}}$.
The algorithm:
    1. $\beta \leftarrow 2$
    2. for $i = 1, \ldots, m$ do
           $\mathbf{w}_i \leftarrow \mathbf{0}$
    3. while $\beta \geq 0.001$ and $Accuracy(\phi_{\mathbf{g}}, D) \leq 0.99$ do
           a. Select an instance $(\mathbf{x}, c_i) \in D$ at random
           b. if $\phi_{\mathbf{g}}(\mathbf{x}) = c_j$ and $j \neq i$ then
                  i. $\mathbf{y} \leftarrow [x_1, \ldots, x_m, 1]^T$
                  ii. $k \leftarrow \frac{(\mathbf{w}_j - \mathbf{w}_i)^T \mathbf{y}}{2\mathbf{y}^T \mathbf{y}}$
                  iii. if $k < \beta$ then
                         A. $c \leftarrow \frac{\beta^2}{\beta + k}$
                         B. $\mathbf{w}_i \leftarrow \mathbf{w}_i + c\mathbf{y}$
                         C. $\mathbf{w}_j \leftarrow \mathbf{w}_j - c\mathbf{y}$
                         D. if the magnitude of $\phi_{\mathbf{g}}$ decreased but increased in previous adjustment
                            then
                                $\beta \leftarrow a\beta - b$
    4. return $\phi_{\mathbf{g}}$
```

The term *magnitude of* $\phi_{\mathbf{g}}$, referred to in item 3(b)iiiD of the algorithm, means the sum of magnitudes of the weights $\mathbf{w}_i$, $i = 1, \ldots, m$.

If the problem is linearly separable, then the procedure will find a solution in a finite time (Duda et al 2001). Otherwise, error corrections would not end. Hence the parameter β has been introduced to realize the idea of *thermal perceptron* (Frean 1990). It is reduced from time to time to simulate the annealing phenomenon, which guarantees convergence of the process also in the case of linearly non-separable data.

Variable k is set to the *absolute error correction* needed to correctly classify the misclassified object. But it is not used directly in weight change formulae. To eliminate deterioration in the convergence process caused by error correction for the cases of misclassified instances located very far or very close to the decision border, the c parameter controlling weight changes is set to $\frac{\beta^2}{\beta+k}$, which guarantees process stability.

To help the thermal linear machine separate the classes, the training data objects should be appropriately prepared. Utgoff and Brodley (1991) proposed to standardize numeric features before the process and convert symbolic features to a number of binary features. If the feature has just two possible symbols, they can be encoded as +1 and -1 respectively. Otherwise, the symbolic attribute is encoded by a number of binary features equal to the number of possible symbols. For a given data object, one of the new features (the one corresponding to the symbol assigned to the object) gets the value of +1 and all the remaining features get -1. Missing values are replaced by 0s, which correspond to the means of the values observed in the training sample.

It may happen that a linear machine does not in fact split the node—all training data objects belong to the same part of the feature space split. In such cases the node gets closed as a leaf, regardless of the fact that it is not pure.

Minimization of arbitrary misclassification cost functions was introduced to the LMDT approach by Draper et al. (1994). They assigned proportions to the classes (all equal to 1 at start) to reflect the misclassification costs and respect them in the thermal learning of the linear machine.

Variable Elimination

To make the models of LMDT as simple as possible, and sometimes more accurate, Utgoff and Brodley (1991) also described a technique to eliminate variables during the learning processes. They proposed to repeat training after elimination of the feature that contributes least to the discrimination. The best solutions, found so far, must be recorded and new ones compared to them so as to estimate if the results do not deteriorate. The comparisons performed by Utgoff and Brodley (1991) used t-test with $\alpha = 0.01$ to estimate if the new results are not significantly worse than the saved result. Moreover, they used an additional threshold parameter defining the size of acceptable decline in accuracy when the feature is eliminated.

After a number of train and eliminate cycles, the best result, saved in appropriate time, is returned as the final linear machine. The scenario described here is the strategy of *sequential backward elimination* (one of the fundamental approaches to feature selection). Brodley and Utgoff (1992a) have also proposed an approach of *sequential forward selection*, where the best single feature is selected first, and then subsequent features are added one by one, so as to maximize the increase of a merit criterion.

Moreover, they have suggested a combination of the two strategies, referred to as *heuristic sequential search*. Its idea is to run the first stages of both approaches (Forward selection and Backward elimination) and select the better of the two. Such initial test may detect, whether there are many noise features that spoil the result or most of the features are important, and select the method accordingly.

2.3.2 OC1

Oblique Classifier 1 (OC1, Murthy et al. 1993; Murthy et al 1994; Murthy 1997) is a method for DT construction by means of a search for optimal hyperplane separating classes of objects. The search uses a heuristics to find local minima and ideas of non-deterministic approaches to get out of the minima in the pursuit of better solutions.

At each DT node a single hyperplane is determined, so the resulting trees are binary. Similarly to the LMDT approach, the hyperplanes are defined by $m + 1$-dimensional vectors, where m is the dimensionality of the object space. The key procedure of OC1 is its, so called, *perturbation method*, adjusting one selected coefficient in the hyperplane to maximize a measure of impurity of the hyperplane split.

Algorithm 2.8 (OC1 hyperplane perturbation algorithm)

Prototype: ***OC1Perturb(w,d,D,Impurity)***

Input: *Initial hyperplane parameters* $\mathbf{w} = (w_1, \ldots, w_{m+1})$*, index of the dimension to be perturbed d, training dataset* $D \subseteq O \times \mathscr{C}$ $(O \subseteq R^m)$*,* $D = \{(\mathbf{x}_i, c_i), i = 1, \ldots, n\}$*, hyperplane split impurity measure Impurity.*

Output: *Modified hyperplane parameters* $\mathbf{w}$.

The algorithm:

```
1. for $i = 1, \ldots, n$ do
      a. $V_i \leftarrow \sum_{j=1}^{m} w_j x_j + w_{m+1}$
      b. $U_i \leftarrow \frac{w_d x_{id} - V_i}{x_{id}}$
2. Sort $U_1, \ldots, U_n$ in nondecreasing order
3. $\mathbf{w}' \leftarrow \mathbf{w}$
4. $w'_d \leftarrow$ the best 1-D split of the sorted $U_1, \ldots, U_n$
5. if $Impurity(\mathbf{w}, D) < Impurity(\mathbf{w}', D)$ then
      a. $w_d \leftarrow w'_d$
      b. $stagnant \leftarrow 0$
   else if $Impurity(\mathbf{w}, D) = Impurity(\mathbf{w}', D)$ then
      a. $w_d \leftarrow w'_d$ with probability $e^{-stagnant}$
      b. $stagnant \leftarrow stagnant + 1$
6. return $\mathbf{w}$
```

The procedure is presented as Algorithm 2.8. It calculates U_i values for each data object, sorts the objects by nondecreasing values of U_i and performs linear search to find the best 1-D split of the data.

Murthy et al. (1993) performed the hyperplane perturbations in three different ways:

Seq: The coefficients were perturbed one by one in sequence, many times, until none of the coefficient values were modified:

```
Repeat
        1. $\mathbf{v} \leftarrow \mathbf{w}$
        2. for $i = 1, \ldots, m + 1$ do
               $\mathbf{w} \leftarrow$ OC1Perturb($\mathbf{w}$,i,D,Impurity)
until $\mathbf{v} = \mathbf{w}$
```

Best: Each coefficient was perturbed independently, to get the one providing maximum impurity reduction. The optimization was run until the same coefficient was returned twice in sequence:

```
Repeat
        1. $j \leftarrow$ the coefficient providing maximum impurity reduction when perturbed
        2. OC1Perturb($\mathbf{w}$,j,D,Impurity)
until $j$ is the same as in the previous iteration
```

R-50: The coefficient to be perturbed was selected randomly 50 times:

```
for $i = 1, \ldots, 50$ do
        1. $j \leftarrow$ random integer between $1$ and $m + 1$
        2. OC1Perturb($\mathbf{w}$,j,D,Impurity)
```

Escaping from Local Minima

The procedures, described above, optimize hyperplanes in such a way that when a local minimum is reached, no further perturbation reduces the impurity. To increase the probability of finding global minimum, two techniques were applied by Murthy et al. (1993): perturbing coefficients in a random direction and choosing multiple initial hyperplanes. In the first technique, they selected randomly a vector $\mathbf{r} = (r_1, \ldots, r_{m+1})$ and analyzed hyperplanes determined by vectors of the form $\mathbf{w} + \alpha\mathbf{r}$. In a perturbation procedure analogous to the algorithm 2.8, they calculated the best value of α from the point of view of hyperplane impurity. If the new hyperplane impurity was lower than that of $\mathbf{w}$, the perturbation procedure was continued for the new coefficients. Otherwise, the hyperplane of $\mathbf{w}$ was returned as the final result.

The other method was just to start with different initial hyperplane vectors $\mathbf{w}$ and select the best of the local minima found with the OC1 optimization procedure.

Hyperplane Impurity Measures

Algorithm 2.8 is parameterized by the method to calculate impurity of a hyperplane in the context of particular dataset. Murthy et al. (1993) have proposed three methods to measure such impurity:

max minority : $MM(s, D) = \max(\min(|D_{s_1,c_1}|, |D_{s_1,c_2}|), \min(|D_{s_2,c_1}|, |D_{s_2,c_2}|))$,
sum minority : $SM(s, D) = \sum_{i=1}^{2} \min(|D_{s_i,c_1}|, |D_{s_i,c_2}|)$,
sum of impurity : $SI(s, D) = \sum_{i=1}^{2} \sum_{(\mathbf{x},c)\in D_{s_i}} (Bin(c) - avg_i)^2$, where $Bin(c) \in \{0, 1\}$ is a binary coding of the classes and $avg_i = \frac{1}{|D_{s_i}|} \sum_{(\mathbf{x},c)\in D_{s_i}} Bin(c)$.

Although the measures are defined for split s in general, while Algorithm 2.8 calls the *Impurity* method with vectors determining hyperplanes, it is easy to make them compatible also formally, as the hyperplane splits the space into halves. Also, it is not difficult to extend the definitions given by Murthy et al. (1993) to arbitrary numbers of classes.

Other Solutions

Murthy et al (1994) have listed additional measures like information gain, Gini index or twoing criterion. In fact, any measure of split quality may be used here.

Similarly, any pruning method is suitable for OC1 trees, but Murthy et al (1994) have used the cost-complexity optimization cost-complexity pruning of Breiman et al. (1984).

In the original definition, missing values in the data were replaced by mean values of the attribute, before training or testing OC1.

2.3.3 LTree, QTree and LgTree

Probabilistic linear trees LTree Gama(1997) result from a combination of three ideas: *divide and conquer* methodology of decision trees, *linear discriminant analysis* and *constructive induction*.

According to the paradigm of constructive induction, each split of an LTree node (the corresponding training data sample) is performed in two independent steps:

- new attributes construction (linear combinations of existing features),
- best split determination by a technique for univariate DT construction.

Such scheme and other ideas introduced to LTree induction caused important differences between the approach and other methods, introduced earlier:

- the number of features describing data can differ between the nodes of the same classification tree,
- new attributes, once created, are propagated downwards, so that other splits in the branch can also use them,
- the trees estimate probabilities by an analysis of data distributions in the whole path followed when classifying an object: from the root to appropriate leaf.

Attribute Construction

When new features are generated for the sake of a tree node split, it is important that it brings as much information about class discrimination as possible. To determine the class of a given object $\mathbf{x}$, it is sufficient to determine conditional probabilities $P(c|\mathbf{x})$ for each possible class c. The most probable class should be indicated as the class of $\mathbf{x}$. Given the Bayes theorem:

$$P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})}. \tag{2.45}$$

To determine the winner class, the denominator (common for all the classes) can be ignored leaving $P(c)P(\mathbf{x}|c)$ as the value to be estimated for each class $c \in \mathscr{C}$.

Linear discriminant analysis assumes that each class is normally distributed and that all classes share the covariance matrix Σ. In such case, maximization of $P(c)P(\mathbf{x}|c)$ is equivalent to maximization of $P(c)f_c(\mathbf{x})$, where $f_c(\mathbf{x})$ is the probability density function for class c. Multidimensional normal density function of mean μ and covariance Σ is given by:

$$f_{\mu,\Sigma}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)}. \tag{2.46}$$

After some more simplifications (like comparing logarithms instead of exponential expressions and throwing out the constant components for all classes) one can easily get:

$$P(c|\mathbf{x}) \propto \log(P(c)) + \mathbf{x}^T\Sigma^{-1}\mu_c - \frac{1}{2}\mu_c^T\Sigma^{-1}\mu_c. \tag{2.47}$$

Therefore, Gama(1997) defined the linear discriminant hyperplane as:

$$H_c = \alpha_c + \mathbf{x}^T\beta_c, \tag{2.48}$$

where $\alpha_c = \log(P(c)) - \frac{1}{2}\mu^T\Sigma^{-1}\mathbf{x}$ and $\beta_c = \Sigma^{-1}\mu_c$. The mean μ_c and covariance matrix Σ are calculated as the training sample mean and pooled covariance matrix. Because in some circumstances the polled covariance matrix may be singular, Gama (1999) suggested using SVD (see appendix section B.2) to reduce the features that cause the singularity.

The hyperplane formula (2.48) defines new features added at the tree node. New features are constructed for $k_N - 1$ classes of the k_N represented in the node N by the number of objects greater than the number of attributes (Gama (1999) suggested twice the number of attributes).

When adding the attributes, the α_c is constant for all the objects and could be omitted. In fact, the fragments ignored when deriving (2.47) as not important from the point of view of winner-class selection for an object are not meaningless when comparing different objects and could play significant role here.

Gama (1999) also proposed two other discriminant approaches to be used in analogous ways as the linear one: *quadratic* and *logistic discriminants*, yielding algorithms named *QTree* and *LgTree* respectively. Quadratic discriminants can be inferred similarly to the linear ones from the assumption of normal distributions but without the constraint of equal covariance matrices for all the classes. Similar reasoning as for linear discriminants brings the conclusion that

$$P(c|\mathbf{x}) \propto \log(P(c)) - \frac{1}{2}\log(|\Sigma_c|) - \frac{1}{2}(\mathbf{x}-\mu_c)^T\Sigma_c^{-1}(\mathbf{x}-\mu_c), \tag{2.49}$$

where Σ_c is the covariance matrix for class c. Again, possible problems with covariance matrix singularity can be solved by SVD analysis and feature elimination.

In the LgTree algorithm, generation of new features starts with selecting one of the classes $(c_1, \ldots, c_k)$ as so called *reference class* (say the last one c_k) and $k-1$ vectors $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{k-1}$ to estimate the conditional class probabilities:

$$P(c_k|\mathbf{x}) = \frac{1}{\sum_{j=1}^{k-1} e^{\boldsymbol{\beta}_j \mathbf{x}}}, \quad P(c_i|\mathbf{x}) = \frac{e^{\boldsymbol{\beta}_i \mathbf{x}}}{\sum_{j=1}^{k-1} e^{\boldsymbol{\beta}_j \mathbf{x}}}, \quad i = 1, \ldots, k-1. \tag{2.50}$$

Then the Newton-Raphson iterative procedure is used to find such βs that maximize the likelihood:

$$L(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{k-1}) = \prod_{i=1}^{k} \prod_{\mathbf{x} \in D_{c_i}} P(c_i|\mathbf{x}). \tag{2.51}$$

As in the case of linear discriminants, the subsequent $\boldsymbol{\beta}$s can be used as the projection vectors to create new features discriminating the classes.

Split Criteria

The constructive approach can be combined with any method of univariate DT induction. By default Gama (1997, 1999) used a method very similar to C4.5: the Information gain criterion was used for split quality estimation. Continuous features splits were binary, while using categorical features for splits resulted in multi-way branching (as many subnodes as symbols of the feature).

Decision Making

As mentioned above, the LTree family methods have implemented a special way of class probability estimation, based on data distribution in the whole path from the root to the leaf of interest, instead of the most common solution to base just on the distribution in the leaf. Gama (1997) suggested weighting class proportions in a given node N with its parents probabilities:

$$P(c|N) = \frac{P(c|Parent(N)) + w\frac{n_{N,c}}{n_N}}{1+w}. \tag{2.52}$$

Only for the root node, class probabilities were based directly on class frequencies in the training data sample. The parameter w was set to 1 by default, but its value was up to the user. Such definition of class probabilities may cause that the winner class is not the majority class of the leaf.

Pruning

To obtain well generalizing trees, Gama (1997) suggested using one of two methods: the Error-Based Pruning of C4.5 (but without the option of grafting the best subtree in the place of its parent, when the subtree seems to be better) and another one, strongly related to the way of calculating probabilities, described above. When the sum of errors made by subbranches of a given node, was not less than the error of the (parent) node acting as a leaf, then the node was turned into a leaf. In the most common approach of estimating probabilities by leaves frequencies, it cannot happen that the children nodes make, in total, more errors than their parent, but it is possible with so untypical probability estimation, as described above.

Algorithm 2.9 (Iterative refiltering)

Prototype: ***IterativeRefiltering(D,LM)***
Input: *Training dataset D, a learning machine LM.*
Output: *A model.*
The algorithm:

```
1. repeat
      a. $M \leftarrow Train(LM, D)$ /* model M is the result of training LM on D */
      b. $D' \leftarrow Classify(M, D)$ /* $D'$ is D with classes predicted by the ones provided by
         M */
      c. $D \leftarrow D \cap D'$
   while $D' \neq D$ do
2. return $M$
```

2.3.4 DT-SE, DT-SEP and DT-SEPIR

John (1995b, 1996) has contributed to the field of multivariate DT induction by presenting the ideas of using soft criteria for split quality estimation and iterative refiltering in DT regularization. Soft miltidimensional criteria are not compatible with symbolic features, so the family of DT-SE methods needs some data preprocessing to get rid of the symbols which have no sensible order. In the experiments of John (1996), all unordered categorical attributes were converted into corresponding 0-1 indicator variables (see p.6) to prevent introduction of accidental information or hiding existing information by arbitrary symbols ordering. Therefore, binary splits are most natural in this family of trees, although one could easily apply the same optimization methods for analyzing multi-way splits.

Soft Split Criteria and Soft Entropy

In general, the proposition of John (1996) was softening different criteria evaluating split quality, so that they can be optimized with methods like gradient descent. To make it easier, he described the problem of finding an optimal split of a dataset D as the problem of finding parameters $\theta^\star \in \Theta$ of a split function g_θ that minimize some split quality measure I:

$$\theta^\star = \arg\min_{\theta \in \Theta} I(D|g_\theta). \tag{2.53}$$

The domain Θ can be arbitrarily defined for particular problems: it may contain simple, single-value parameters like split thresholds, but also more sophisticated objects consisting of more important values parameterizing the split function g_θ.

The objective function I may be any split criterion including information gain, Gini index and so on. Although most of the criteria are defined on the basis of cardinalities of datasets $D \in O$, they can be easily softened by using fuzzy membership functions and fuzzy cardinalities: $|D|_w = \sum_{x \in O} w(x)$. An example of a fuzzy membership function is the sigmoid function performing linear discriminant splitting:

$$g_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}. \tag{2.54}$$

The composition of the objective function from a split function like (2.54) and split quality measure I helps in using gradient methods for minimization. By the chain rule $\frac{\delta I}{\delta \theta} = \frac{\delta I}{\delta g_\theta} \frac{\delta g_\theta}{\delta \theta}$, so if the quality measure function is differentiable with respect to g_θ and g_θ with respect to θ, performing gradient descent minimization is very easy.

The experiments of John (1996) were performed with the information gain function as the split quality measure and the split function g_θ defined by (2.54). More precisely, the negated sum of entropies (2.5) of the datasets resulting from the split needs to be calculated, as the entropy of the whole dataset is constant for all splits. The result got the name of *soft entropy* (SE) which gave rise to the name DT-SE. A simple steepest descent procedure was used for objective function minimization. After the $\theta^\star$ was found, the dataset was crisply split into two parts corresponding to the conditions $g_{\theta^\star}(\mathbf{x}) \geq \frac{1}{2}$ and $g_{\theta^\star}(\mathbf{x}) < \frac{1}{2}$ which are equivalent to a typical linear discriminant solution: $\theta^{\star T} \mathbf{x} \geq 0$ or $\theta^{\star T} \mathbf{x} < 0$.

Pruning

Tree regularization process prepared by John (1996) for his DT-SE trees used two kinds of methods: pre-pruning and retraining. The former is very simple: tree nodes are not further split if one of the classes has less than 5 representatives. Naturally, the 5 is the value of a parameter, but in John's experiments it was set just to 5. The technique of retraining got the name of *iterative refiltering* and was borrowed from the field of regression methods, where it had been used under the names of *robust* (Huber 1977) or *resistant* (Hastie and Tibshirani 1990) *fitting*. A general definition of iterative refiltering is presented as algorithm 2.9. It could be made even more general by replacing the call of *Classify()* method by a more general one and generalizing the stop condition of the loop. The idea behind the algorithm is that if an object is classified incorrectly by the model, then it probably spoils not only the leaves of the tree but also the nodes along the whole path from the root, so it should be advantageous to get rid of such object during learning. The procedure just repeats learning, testing the resulting model and removing the data objects incorrectly classified by the model, until the subsequent model classifies correctly all the objects that remain in the training dataset. The algorithm was presented first by (John 1995a) in application to C4.5 creating the method named *Robust C4.5*. The training procedure called within iterative refiltering must be equipped with some generalization mechanisms. Therefore, in application to DT-SE, the simple stop criterion was used. C4.5 has its own pruning strategy, so it was used for tree regularization.

The DT-SE method augmented by the simple stop-criterion for the purpose of pruning was named DT-SEP and the version using also Iterative refiltering is called DT-SEPIR.

2.3.5 LDT

Yildiz and Alpaydin (2000, 2005a) performed an analysis of six aspects of DT induction algorithms and proposed their own algorithm named *Linear Decision Trees* (LDT). The six aspects are:

- node type: univariate or multivariate,
- branching factor: splitting to two or more subnodes,
- grouping classes into two superclasses for binary trees,
- split quality measures,
- minimization methods for finding best splits.

The resulting LDT algorithm creates binary trees, performing Discriminant analysis at each node (with options of univariate or multivariate splits and Linear discriminant analysis linear or)Quadratic discriminant analysis after the classes with representatives in the node, are grouped into two superclasses with one of two algorithms.

The Discriminant Analysis

LDT follows the idea of *Fisher's linear discriminant analysis* (FDA). The goal of FDA is finding the hyperplane providing the best separation of two groups of objects. In other words, the direction $\mathbf{w}^\star$ that maximizes the distances between different class centers while minimizing the average variance within classes:

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}\in R^k} \frac{w^T \mathbf{S}_B w}{w^T \mathbf{S}_W w}, \tag{2.55}$$

where S_B is the between-class covariance matrix and S_W is the within-class covariance matrix:

$$\mathbf{S}_B = (\overline{x}_{c_1} - \overline{x}_{c_2})(\overline{x}_{c_1} - \overline{x}_{c_2})^T, \tag{2.56}$$

$$\mathbf{S}_W = \sum_{c \in \{c_1, c_2\}} \sum_{\mathbf{x} \in c} (\mathbf{x} - \overline{x}_c)(\mathbf{x} - \overline{x}_c)^T. \tag{2.57}$$

The solution of the maximization problem is

$$\mathbf{w}^\star = \mathbf{S}_W^{-1}(\overline{x}_{c_1} - \overline{x}_{c_2}), \tag{2.58}$$

so the projection $\mathbf{w}^{\star T}\mathbf{x}$ provides optimal class discrimination. Assuming normal distributions of the separated groups, the optimal threshold in the new dimension is

$$w_0 = -\frac{1}{2}(\overline{x}_{c_1} + \overline{x}_{c_2})^T \mathbf{S}_W^{-1}(\overline{x}_{c_1} - \overline{x}_{c_2}) - \log \frac{n_{c_1}}{n_{c_2}}. \tag{2.59}$$

Alternatively, one can analyze all possible split positions and select the best one according to a given quality measure like classification accuracy, entropy, Gini index and so on.

The new dimension created as a linear combination of the original variables describing the data can be handled in the same way as the original features in univariate DTs. The split point w_0 of (2.59) is the optimal split point, on the assumption of equal variances of the new feature in the two groups. When resigning from the assumption about the parameters of normal distributions, the QDA analysis can be done to determine the split point in the same way as in QUEST (see Eqs. (2.17), (2.18), and (2.19)). In the case of equal variances the quadratic equation has one root equivalent to w_0 of (2.59). Otherwise, there can be two roots. If so, the one between the means of the groups (if exists) is selected. If not, or both roots are outside the interval designated by the means, then the middle point between the means is used as the split point.

Symbolic features are transformed to the appropriate number of Indicator variables, and the split is found in such space. As described in the case of QUEST, such splits can be easily decoded back to the original feature and verbalized in the language of the symbolic input feature.

Avoiding Problems of Covariance Matrix Singularity

When a linear dependency exists between two variables describing the data, the matrix $\mathbf{S}_W$ is singular and $\mathbf{S}_W^{-1}$ does not exist. Similarly to the approach of FACT, Yildiz and Alpaydin (2000) propose using *principal component analysis* (PCA) to get rid of the undesirable features. The difference between the two is the way the limitation is introduced. Here, provided the Eigenvalues $\lambda_1, \ldots, \lambda_r$ in nonincreasing order, and the associated Eigenvectors $\mathbf{c}_1, \ldots, \mathbf{c}_r$, minimum d is determined, such that d initial eigenvectors explain more than ε of the variance:

$$\frac{\lambda_1 + \cdots + \lambda_d}{\lambda_1 + \cdots + \lambda_d + \cdots + \lambda_r} > \varepsilon. \tag{2.60}$$

Then, each data vector $\mathbf{x}$ is transformed into the space of selected eigenvectors to

$$\mathbf{z} = (\mathbf{c}_1^T \mathbf{x}, \ldots, \mathbf{c}_d^T \mathbf{x})^T, \tag{2.61}$$

and the LDA is performed in the new space, where the inverse of corresponding $\mathbf{S}_W$ certainly exists. The solution is then transformed back to the original space.

Grouping into Superclasses

When the number k of classes is greater than 2, the LDA procedure described above is preceded by the stage of grouping the classes into two superclasses. Yildiz and Alpaydin (2000) proposed two algorithms for this purpose: one based on *selection* and the other on *exchange*. Unlike the method of QUEST (ClusteringTwo-means clustering), the methods of LDT are supervised. The selection method is presented as Algorithm 2.10. Different random selection at start of the process may generate different superclasses. Yildiz and Alpaydin (2000) suggested to select two most distant classes in place of the random selection.

Algorithm 2.10 (LDT superclasses by selection)

Prototype:* *SuperclassesBySelection(D, SQM)

Input: *Training dataset D with classes* $\mathscr{C} = \{c_1, \ldots, c_k\}$*, a Split quality measure SQM.*

Output: *Superclasses A and B.*

The algorithm:

```
1. Select two classes $c_a$ and $c_b$ from $\mathscr{C}$ at random
2. $A \leftarrow c_a$
3. $B \leftarrow c_b$
4. $\mathscr{C} \leftarrow \mathscr{C} \setminus \{c_a, c_b\}$
5. while $\mathscr{C} \neq \emptyset$ do
     a. $s \leftarrow$ the split provided by the FDA for data $D$ and classes $A$ and $B$.
     b. Select the class $c \in \mathscr{C}$ that added to $A$ or $B$ results in the best result returned by
        SQM
     c. if $c$ maximized SQM when added to $A$ then
          $A \leftarrow A \cup c$
        else
          $B \leftarrow B \cup c$
     d. $\mathscr{C} \leftarrow \mathscr{C} \setminus \{c\}$
6. return $A$ and $B$
```

The second solution offered by LDT is Algorithm 2.11 constructing Superclasses by exchange, proposed by Guo and Gelfand (1992). It divides the classes into two superclasses and then moves the classes from one superclass to the other so as to maximize Information gain of LDA splits. Yildiz and Alpaydin (2000) also suggested a heuristic to replace random initialization: they started with two most distant classes and added remaining ones to appropriate parts, according to the rule of minimum inter-mean distance.

Algorithm 2.11 (LDT superclasses by exchange)

Prototype: SuperclassesByExchange(D)
Input: *Training dataset D with classes $\mathscr{C} = \{c_1, \ldots, c_k\}$.*
Output: *Superclasses A and B.*
The algorithm:

```
1. $A \leftarrow \bigcup_{i \le \frac{k}{2}} c_i, \quad B \leftarrow \bigcup_{i > \frac{k}{2}} c_i$
2. repeat
      a. $s \leftarrow$ the split provided by the FDA for data D and classes A and B
      b. $IG_0 \leftarrow IG(s, D)$ /* information gain – see equation (2.6) */
      c. for $i = 1, \ldots, k$ do
           i. Construct $A_i$ and $B_i$ by moving $c_i$ from its superclass to the other
          ii. $s_i \leftarrow$ the split provided by the FDA for data D and classes $A_i$ and $B_i$
         iii. $IG_i \leftarrow IG(s_i, D)$
      d. $i^\star \leftarrow \arg\max_{i=1,\ldots,k} IG_i$
      e. if $IG_{i^\star} > IG_0$ then
            $A \leftarrow A_{i^\star}, B \leftarrow B_{i^\star}$
   until $IG^\star \le IG_0$ /* no improvement */
3. return A and B
```

Both superclass generation methods require multiple runs of LDA, so they may be a significant additional cost, as they are run for each DT node, generated during the induction process.

2.3.6 Dipolar Criteria for DT Induction

Decision trees based on *dipolar criteria*, offered by Bobrowski and Krętowski (2000), are based on the same fundamental idea as the SSV criterion (see Sect. 2.2.7): to find splits that separate possibly many pairs of objects belonging to different classes. Bobrowski and Krętowski (2000) defined *dipoles* as pairs of objects and distinguished between *pure dipoles* (containing objects belonging to the same class) and *mixed dipoles* (containing objects from different classes).

To define the objective, dipolar criterion function, two penalty functions need to be defined first, for each input vector $\mathbf{v}$, to control whether it stays on the positive or negative side of the candidate hyperplane:

$$\varphi_{\mathbf{v}}^{+}(\mathbf{w}) = \begin{cases} \delta - \mathbf{w}^T\mathbf{v} & \text{if } \mathbf{w}^T\mathbf{v} < \delta, \\ 0 & \text{if } \mathbf{w}^T\mathbf{v} \geq \delta, \end{cases} \tag{2.62}$$

$$\varphi_{\mathbf{v}}^{-}(\mathbf{w}) = \begin{cases} \delta + \mathbf{w}^T\mathbf{v} & \text{if } \mathbf{w}^T\mathbf{v} > -\delta, \\ 0 & \text{if } \mathbf{w}^T\mathbf{v} \leq -\delta, \end{cases} \tag{2.63}$$

where δ is a margin parameter (usually set to 1 by the authors).

Then, for each dipole, the functions can be combined to measure the cost related to dividing the dipole or not. For pure dipoles, the cost can be calculated as

$$\varphi_{\mathbf{u},\mathbf{v}}^{p}(\mathbf{w}) = \varphi_{\mathbf{u}}^{+}(\mathbf{w}) + \varphi_{\mathbf{v}}^{+}(\mathbf{w}) \quad \text{or} \quad \varphi_{\mathbf{u},\mathbf{v}}^{p}(\mathbf{w}) = \varphi_{\mathbf{u}}^{-}(\mathbf{w}) + \varphi_{\mathbf{v}}^{-}(\mathbf{w}), \tag{2.64}$$

while for mixed dipoles as:

$$\varphi_{\mathbf{u},\mathbf{v}}^{m}(\mathbf{w}) = \varphi_{\mathbf{u}}^{+}(\mathbf{w}) + \varphi_{\mathbf{v}}^{-}(\mathbf{w}) \quad \text{or} \quad \varphi_{\mathbf{u},\mathbf{v}}^{m}(\mathbf{w}) = \varphi_{\mathbf{u}}^{-}(\mathbf{w}) + \varphi_{\mathbf{v}}^{+}(\mathbf{w}). \tag{2.65}$$

Eventually, appropriately weighted costs of the dipoles compose the dipolar criterion to be optimized:

$$\Psi(\mathbf{w}) = \sum_{(\mathbf{u},\mathbf{v})\in I_p} \alpha_{\mathbf{u},\mathbf{v}} \varphi_{\mathbf{u},\mathbf{v}}^{p}(\mathbf{w}) + \sum_{(\mathbf{u},\mathbf{v})\in I_m} \alpha_{\mathbf{u},\mathbf{v}} \varphi_{\mathbf{u},\mathbf{v}}^{m}(\mathbf{w}). \tag{2.66}$$

Optimization of this kind of objective functions, can be performed with a Basis exchange algorithm (Bobrowski 1991, 2005) similar to the standard methods of Linear programming. An additional difficulty is the orientation of the dipoles, which makes only one of the two alternative forms of each penalty function (2.64) and (2.65) adequate in particular circumstances. For this purpose, the basis exchange algorithm has been equipped with proper search for adequate orientations of the dipoles (Bobrowski 1999, 2005).

Oblique decision trees stand a chance to preserve some comprehensibility if the linear combinations do not include large numbers of features. Therefore, the DT induction algorithm based on the dipolar criterion, implements some Feature selection functionality. In this aspect, the authors followed Brodley and Utgoff (1992a) and used their Heuristic sequential search of the LMDT (see Sect. 2.3.1). The procedure is the most time consuming part of the overall DT induction process.

A step toward cost decline may be early stopping of the splits. Bobrowski and Krętowski (2000) used a simple rule of stopping the splits, when the count of training data objects falling into the node was less than 5. Apart from that, they pruned the trees after construction, according to the principle of Reduced Error Pruning (see Sect. 2.4.3.1). They used 70 % of the training dataset for tree construction and afterwards, decided which nodes to prune by validating the tree on the dataset containing the remaining 30 % of data.

2.4 Generalization Capabilities of Decision Trees

There are many reasons, for which, decision trees generated on the basis of a training data sample are not perfect classifiers for the whole population of data objects. One of them is imperfectness of the training data sample. It often happens that the data objects descriptions are noisy. The noise may come from imprecise measurements, errors made during data collection, loosing some data and so on. On the other side, very often, the data sample is not representative of the problem and does not contain full information about the relations being learned. Sometimes, the information is contained in the data, but it is too difficult to discover it with the learning processes, because there are many local minima, which “conceal” the global minimum or the decision functions of the learning machines are not capable of describing the hidden dependencies. In all these cases and many others, the model resulting from DT induction may overfit the training data in the sense that it perfectly describes the sample, but not exactly the relations of interest.

When Overfitting occurs close to the root node, then the tree model is completely inaccurate and often, nothing can be done to fix it, but when it is due to splits close to the leaves, Pruning some tree branches can be a successful solution. Pruning makes the models simpler, so it is compliant with the idea of the Occam’s razor. Fortunately, the root node and other nodes close to it, are usually created on the basis of large data samples, and because of that, generalize well. They should not be affected by pruning techniques, which ought to delete relatively small nodes, responsible for distinction between single data items or very small groups, existent in the training sample but being exceptions rather than representative objects of the population.

Another kind of problems can be observed when the number of features describing data objects is large in relation to the number of objects. Then, it is probable that the data contains features accidentally correlated with the output variable. Such features may be selected by the DT induction algorithms as the most informative ones and may significantly distort the model. In such cases, pruning is less helpful—a better way is to create many models and combine their decisions. In the Ensemble, the influence of accidentally correlated features is likely to be dominated by really informative ones. The price to pay for that is often model comprehensibility, but one can still search for some explanations by exploration of the ensemble members with respect to their compatibility with the ensemble decisions, and so on.

Some researchers, for example Bohanec and Bratko (1994) and Almuallim (1996), have used pruning techniques for tree simplification, which they describe as slightly different task than increasing generalization capabilities. They assume that the full tree is maximally accurate and search for simpler descriptions of the data, consciously accepting some loss in accuracy. So the task is to minimize the loss in accuracy for a given size constraint for the tree.

Most of the commonly used pruning techniques belong to one of two groups:

- *pre-pruning*: the methods acting within the process of DT construction, which can block splitting particular nodes,

- *post-pruning*: the methods that act after complete trees are built and prune them afterwards by removing the nodes estimated as not generalizing well.

Some other techniques, aimed at tree generalization, do not fit either of the groups. An interesting example is the strategy of Iterative refiltering used in DT-SE family of methods (see Sect. 2.3.4), where the final trees are results of multi-stage DT induction with adequate adjustment of the training data sample. Yet another (completely different) approach is the optimization of decision rules, but in fact, they are not proper DT pruning methods, as their final results may not have the form of DTs, even when started with the classification rule sets exactly corresponding to DTs.

Pre-pruning methods are also called *stop criteria*. The most natural condition to stop splitting is when no sensible further splits can be found or when the node is *clean* (*pure*), that is, contains objects belonging to only one class. Some generalizations of these ideas are the criteria of stopping when the node is small enough or contains small number of erroneously classified objects (reaching some purity threshold). They also stop further splitting, so are recognized as pre-pruning techniques.

Usually, it is very hard to estimate whether further splitting the node at hand may bring significant information or not, so apart from the simplest conditions like the ones mentioned above, pre-pruning techniques are not commonly used.

Post-pruning methods simplify trees by replacing subtrees by leaves in a previously constructed DT. Again, the group can be divided into two subgroups:

- *direct pruning* methods, that decide which nodes to prune just on the basis of the tree structure and information about training set distribution throughout the tree,
- *validation* methods, that use additional data sample (separate from the one used for training) in a validation process to determine which nodes do not seem to perform well on data unseen during learning.

Among the latter group, there are methods using single validation dataset (like Reduced Error Pruning) and others, performing multiple tests in a process like Cross-validation (for example, Cost-complexity optimization cost-complexity pruning of CART) to estimate optimal values of some parameters to be used in final tree pruning.

2.4.1 Stop Criteria

Like all recursive algorithms, DT induction methods must define a condition to stop further recursion, to avoid infinite loops.

One of the most natural criteria breaking the recursive splitting process is the condition of nodes purity. A clean node, that is, containing objects of one class only, does not need further splits, as it is 100 % correct (as far as the training data is concerned). Some softened purity conditions may also be defined. It is a popular approach to define the maximum allowable number of errors to be made by a tree node. When a node contains objects from one class and n objects from other classes it is not split when $n < \theta$, where θ is a pre-defined threshold. Another variant of this idea is to define the threshold as the proportion of objects from classes other than the majority one, and calculate not just the number of errors, but the percentage of errors yielding different allowable error count for nodes of different sizes. Yet another similar methods put size constraints on each node and do not accept splits that generate a subnode smaller than a given threshold.

Another situation, when the recursive process is stopped in a natural way, is when the split criterion being used does not return any splits for the node. For example, when all the object descriptions are the same, but some data objects belong to one class and some to others. Such circumstances may occur when the dataset is noisy or when the features describing data are not sufficient to distinguish the classes.

Some split criteria can return no splits not only when all the data vectors are the same, but because of the setting of their parameters. It is common in the split methods based on statistical tests, that a split is acceptable only if a statistical test rejects the hypothesis of concern, with a specified level of confidence. For example, algorithms like Cal5 or CTree test some hypothesis to determine the split feature and the split of the node. On the basis of the tests they make decisions whether to make further splits or not. They may decide to not split the node even if it is not pure. Naturally, such statistical stop conditions could also be used with algorithms, where the split points are searched with exhaustive methods, but it is not common to do so. It is more popular to use statistical methods to prune the tree after it is fully grown, but such approaches perform post-pruning (the direct part) not pre-pruning.

The authors of DT algorithms usually prefer to build oversized trees and then prune them instead of using advanced stop criteria, because their wrong decisions may significantly affect the quality of the resulting trees. Comparative tests of pre-pruning and post-pruning algorithms (for example, the ones made by Mingers (1989a)) prove that the latter are more efficient, so most often, DT induction algorithms use only the most natural stop criteria. Sometimes, to avoid loosing time for building too big trees, small impurity thresholds are accepted. For example, when a node contains a single object of another class than that of all the others, it is not very reasonable to expect that further splits can generalize well, so ignoring such impurities, in most cases, just saves some time with no negative impact on resulting tree quality.

2.4.2 Direct Pruning Methods

Direct pruning is very time-efficient because it just examines the decision tree and training data distribution throughout the tree. On the other hand, they are provided with information about training data distribution, so they get less information than Validation methods, which are given also the results for some Unseen data. Hence, the task of direct pruning may be regarded as more difficult than the task of validation.

Methods for direct pruning usually estimate misclassification risk of each node and the whole subtree suspended at the node. If the predicted error rate for the node acting as a leaf is not greater than corresponding error for the subtree, than the subtree is replaced by a leaf. The differences between methods of this group are mainly in the way of the misclassification rate estimation, as the natural estimation on the basis of the training data is overoptimistic and leads to oversized trees.

The following subsections describe some direct pruning methods. To provide a valuable review of this family of algorithms, several methods of diverse nature have been selected (including the most popular ones):

- PEP: *Pessimistic Error Pruning* (Quinlan 1987; Mingers 1989a; Esposito et al. 1997),
- EBP: *Error-Based Pruning* (Quinlan 1987; Esposito et al. 1997),
- MEP and MEP2: *Minimum Error Pruning* (Niblett and Bratko 1986; Mingers 1989a; Cestnik and Bratko 1991),
- MDLP– *Minimum Description Length Pruning* – different approaches have been proposed by Quinlan and Rivest (1989), Wallace and Patrick (1993), Mehta et al. (1995), Oliveira et al. (1996), Kononenko (1998); here only the approach of Kononenko (1998) is presented in detail.

2.4.2.1 Pessimistic Error Pruning

PEP Sometimes it is advantageous to approximate binomial distribution with normal distribution. To avoid some unfavorable consequences of the transfer from discrete values to continuous functions, an idea of *continuity correction* has been successfully used (Snedecor and Cochran 1989). Quinlan (1987) proposed to apply it to estimation of the real misclassification rates of DTs. Given a node N with the information about the number n_N of training data objects falling into it and the number e_N of errors (training data items belonging to classes different than the one with majority representation in N), Quinlan estimated the misclassification risk as

$$\frac{e_N + \frac{1}{2}}{n_N}. \tag{2.67}$$

The rate for the subtree T_N rooted at N can be defined as the weighted sum of the rates for all leaves in T_N ($\widetilde{T_N}$) with weights proportional to leaves sizes, which can be simplified to:

$$\frac{\sum_{L \in \widetilde{T_N}} e_L + \frac{1}{2}|\widetilde{T_N}|}{n_N}. \tag{2.68}$$

Because the continuity correction is often not satisfactory to eliminate overoptimistic estimates, the approach of Pessimistic Error Pruning uses it with the margin of one standard deviation of the error (SE for standard error) (Quinlan 1987; Mingers 1989a; Esposito et al. 1997). Denoting the corrected error of the subtree T_N, that is, the numerator of (2.68), by E_{T_N}, the estimation of the standard error can be noted as:

$$SE(E_{T_N}) = \sqrt{\frac{E_{T_N} \times (n_N - E_{T_N})}{n_N}}. \tag{2.69}$$

The algorithm of PEP replaces a subtree by a leave when the error estimated for the node (2.67) is not greater than the corrected error counted for the subtree (2.68) minus its standard error (2.69).

The procedure is applied in top-down manner, which usually eliminates a part of calculations, because if a node at high level is pruned, all the nodes of its subtree need not be examined.

2.4.2.2 Error-Based Pruning

Another way of pessimistic error evaluation gave rise to the Error-Based Pruning algorithm (Quinlan 1987) used in the popular C4.5 algorithm. Although it is often described as PEP augmented with possibility of grafting maximum child in place of the parent node, the difference is much larger—estimation of the pessimistic error is done in completely different way. Here, confidence intervals are calculated for given probability of misclassification and the upper limits of the error rates are compared (for given node as a leaf and the subtree).

From the source code of C4.5r8 it can be discovered that the Wilson's approach to confidence intervals (Wilson 1927) was applied. Probably, the reason for such a choice was that the Wilson's intervals offer good approximation even for small samples, which are very common in DT induction. Actually, pruning is required almost only for nodes with small data samples. Nodes with large samples usually allow for reliable splits and are not pruned.

Wilson defined the confidence interval at level α for a sample of size n, drawn from binomial distribution with probability p as:

$$\frac{p + \frac{z^2}{2n} \pm z\sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}, \tag{2.70}$$

where $z = z_{1-\frac{\alpha}{2}}$ is the critical value of the Normal distribution for confidence level α.

To determine whether to prune a node N or to keep the subtree, or to graft maximum child node in place of N, one needs to calculate the upper limit of the confidence interval for misclassification rate of N as a leaf, the subtree T_N and for the maximum child of N. The decision depends on the fact, which of the three limits is the smallest.

To avoid comparing the quality of the node at hand with all possible results of pruning its subtree, one can compare just to the best possible shape of the subtree. However, to obtain the best pruning of the subtree before its root node pruning is considered, the process must be run from bottom to the root of the tree (as opposed to PEP, which is a top-down approach).

EBP has been commented as more pessimistic than PEP. Esposito et al. (1997) argument that although the estimation is more pessimistic, it is so for both the subtree and its root acting as a leave, which makes the method prune less than the PEP. As can be seen in the experiments described in Chap. 5, on average, EBP generates smaller trees than PEP. The discussion there gives some explanation about the most probable reasons of so different conclusions.

The code of C4.5 contains additional tests, controlling the size of result trees. For example, it checks whether the node to be split contains at least 4 known values (more precisely 2·MINOBJS, where MINOBJS is a parameter with default value of 2) and whether the split would not introduce too small nodes (of size less than MINOBJS). When any of the tests reports the danger of too small DT nodes, the node is not split (gets closed, converted into a leaf).

Existence of missing values in the data is reflected in the method by weights assigned to each training data object. Initially each object gets the weight of 1 and in the case of uncertain splits (splits using feature values not available for the object) the object is passed down to all the subnodes, but with properly decreased weights (equal to the proportions of the training data without missing values, passed down to the subnodes). The numbers of objects in each node are calculated as the sums of weights assigned to objects instead of the crisp counts.

2.4.2.3 Minimum Error Pruning

MEPAs noticed by Niblett and Bratko (1986), misclassification probability estimates of DT leaves can be calculated according to the *Laplace's law of succession* (also called *Laplace correction*). In the case of a classification problem with k classes $c_1, \ldots, c_k$, the class probability distribution may be estimated by:

$$p(c_i) = \frac{n_{N,c_i} + 1}{n_N + k}. \tag{2.71}$$

Cestnik and Bratko (1991) proposed using a more general Bayesian method for estimating probabilities (Good 1965; Berger 1985). According to this method, called *m-probability-estimation*, the estimates (called *m-estimates*) are:

$$p(c_i) = \frac{n_{N,c_i} + p_a(c_i) \cdot m}{n_N + m}, \tag{2.72}$$

where $p_a(c_i)$ is a priori probability of class c_i and m is a parameter of the method.

It is easy to see that the Laplace correction is a special case of m-estimates, where $m = k$ and prior probabilities are all equal to $\frac{1}{k}$. The m parameter serves as a coefficient determining how much the raw training data estimations should be pushed towards a priori probabilities — with $m = 0$ the raw proportions are effective and with $m \to \infty$ the probabilities become the priors.

Given the probability estimation scheme (2.71) or (2.72), the decisions about pruning a node are made according to the results of comparison between the probability of misclassification by the node acting as a leaf, and by the subtree rooted in the node. Such pruning methods are referred to as *MEP* (*Minimum Error Pruning*) and *MEP2* respectively.

To sum up the differences between MEP and MEP2, it must be mentioned that:

- MEP assumes uniform initial distribution of classes, while MEP2 incorporates prior probabilities in error estimation.
- MEP is parameterless and the degree of MEP2 pruning can be controlled with m parameter.
- the m parameter of MEP2 can reduce the influence of the number of classes to the degree of pruning.

It is not obvious what value of m should be used in particular application. Cestnik and Bratko (1991) suggested using domain expert knowledge to define m on the basis of the level of noise in the domain data (the more noise the larger m) or performing validation on a single separate dataset or in the form similar to the one proposed by Breiman et al. (1984) and presented in more detail in Sect. 2.4.3.2. More discussion on validation of the m parameter is given in Sect. 3.2.4.3.

2.4.2.4 Minimum Description Length Pruning

Many different approaches to DT pruning based on the *Minimum Description Length* (MDL) principle have been proposed (Quinlan and Rivest 1989; Wallace and Patrick 1993; Mehta et al. 1995; Oliveira et al. 1996; Kononenko 1998). All of them share the idea that the best classification tree built for a given dataset is the one that offers minimum length of the description of class label assignment for training data objects. Such approaches deal with a trade-off between the size of the tree and the number of exceptions from tree decisions. A nice illustration of the problem is the analogy presented by Kononenko (1995) of the need to transmit data labels from a sender to a receiver with as short message as possible. Naturally, the smaller tree the shorter its encoding, but also the larger part of the training data is misclassified with the tree, so the exceptions require additional bits of code. In other words, if a tree leaf contains objects from one class, it can be very shortly described by the code of the class, and if there are objects from many classes in the leaf, than the class assignment for all the objects must be encoded, resulting in significantly longer description.

The MDL-based DT pruning algorithm implemented and tested in Chap. 5 is the one presented by Kononenko (1998), based on the ideas of using MDL for attribute selection, assuming that the best attribute is the most compressive one (Kononenko 1995).

Given a decision tree node N containing n_N training data objects (belonging to classes $c_1, \ldots, c_k$), the encoding length of the classification of all instances of N can be calculated as:

$$\mathit{PriorMDL}(N) = \log \binom{n_N}{n_{N,c_1}, \ldots, n_{N,c_k}} + \log \binom{n_N + k - 1}{k - 1}. \tag{2.73}$$

The first term represents the encoding length of classes of the n_N instances and the second term represents the encoding length of the class frequency distribution.

The value of $\mathit{PriorMDL}(N)$ suffices for the estimation of the description length of the classification in node N treated as a leaf. The description of the subtree T_N must include the description of the structure of the subtree and classification in all its leaves. After some simplifications, Kononenko (1998) proposed:

$$\mathit{PostMDL}(T_N) = \begin{cases} \mathit{PriorMDL}(N) & \text{if } N \text{ is a leaf} \\ 1 + \sum_{M \in \mathit{Children(N)}} \mathit{PostMDL}(M) & \text{if } N \text{ has children nodes,} \end{cases} \tag{2.74}$$

where $\mathit{Children}(N)$ is the set of children nodes of N.

Eventually, to decide whether to prune at node N or not, it suffices to compare the description lengths of the leaf N and the subtree T_N and prune when

$$\mathit{PriorMDL}(N) < \mathit{PostMDL}(T_N). \tag{2.75}$$

The condition must be tested in bottom-up manner for all nodes of the tree, similarly to EBP and MEP methods.

2.4.2.5 Depth Impurity Pruning

Another interesting approach to decision tree pruning was presented by Fournier and Crémilleux (2002). They defined a measure of DT quality to reflect both purity of DT leaves and DT structure.

An important part of the quality index is *Impurity Quality* of a node N in a tree T:

$$\mathit{IQN}_T(N) = (1 - \varphi(N))\beta^{depth_T(N)-1}, \tag{2.76}$$

where φ is an impurity measure normalized to [0, 1]. Since the node quality (2.76) reflects how deep the node occurs in the tree, *Depth Impurity* of the tree can be defined just as the weighted average of the quality values of all its leaves:

$$\mathit{DI}(T) = \sum_{N \in \widetilde{T}} \frac{n_N}{n_{N_0}} \mathit{IQN}_T(N), \tag{2.77}$$

where N_0 is the root node of T.

The definition (2.77) refers to the whole final tree—the depth in the tree must always be calculated for the whole tree, which is often impractical, because when pruning a subtree, it is usually more reasonable to focus just on the subtree vs its root node. Hence, the DI index can be redefined in a recursive form as:

$$\mathit{DI}(T_N) = \begin{cases} 1 - \varphi(N) & \text{if depth of } T_N \text{ is } 1, \\ \beta \sum_{M \in \mathit{Children}(N)} \frac{n_M}{n_N} \mathit{DI}(T_M) & \text{otherwise.} \end{cases} \tag{2.78}$$

The method of *Depth Impurity Pruning* compares $DI(T_N)$ regarding the full subtree T_N and reduced to a single leaf N. It prunes the node if the *DI* index for the leaf N is lower than the one for the subtree rooted at N.

As usual, proper selection of the β parameter is a nontrivial task. There is no justification for regarding a single value as the best one, so it needs to be determined in special processes, for example in CV based analysis, as described in Sect. 3.2.4.3.

2.4.3 Validation Based Pruning

Validation is a stage of learning, where models being already results of some initial learning processes are adjusted with respect to the results obtained for a data sample different than the one used for the initial learning. It seems a very attractive way of improving DT generalization capabilities, so it has been applied by many authors.

Among numerous DT validation methods, we can distinguish the group of algorithms that perform only a single initial learning followed by a single validation pass and the group of those optimizing parameters in multistage processes.

A typical algorithm belonging to the former group is Reduced Error Pruning (REP), which adjusts the tree to provide minimum classification error for given validation data. The methods based on single training and validation have a disadvantage that the resulting DT model is built on a part of the whole training data (another part is used for validation, so can not be included in tree construction). Therefore, the methods from the latter group have more possibilities in providing accurate trees. Naturally, the methods capable of multi-pass validation are also eligible for single pass validation, but not inversely.

Several validation based pruning methods are presented in the following subsections. Apart from REP, all other methods presented below use Cross-validation to learn how to prune the final tree built on the whole training data.

2.4.3.1 Reduced Error Pruning

The most natural use of a validation dataset to adjust the tree trained on another set is to prune each node, if only it does not increase the classification error calculated for the validation data. Although the method is called *Reduced Error Pruning* (REP), it is advisable to prune nodes also when the error after pruning does not change. According to Occam's razor, a simpler model should be preferred, if it provides the same accuracy as a more complex one. The algorithm passes the validation dataset through the tree to determine numbers of errors made by each node, and analyzes all the splits in the tree, starting from those with leaves as subnodes, up to the root node, by comparing the numbers of errors of the node and the subtree rooted at the node. When the error count of the subtree is not lower than that of its root node, then it is replaced by a leaf.

A theoretical analysis of why the algorithm often fails to prune the tree, although large trees are not significantly more accurate than small ones, was conducted by Oates and Jensen (1999). As a result, they proposed to decide about pruning a node and its subnodes on the basis of different validation samples. It is easy to do so for artificial data, if a new sample can always be generated, but not often feasible in real applications, where data samples are limited, and sometimes so small that even extracting a single validation sample can significantly reduce learning gains.

2.4.3.2 Cost-Complexity Minimization

Accepting Occam's razor in the realm of DT induction implies care for as small trees as possible without decline of models accuracy. This leads to a typical trade-off condition, because pruning branches of trees built for given training data causes deterioration of reclassification scores. Breiman et al. (1984) proposed to control the trade-off with α parameter in a measure of DT misclassification cost involving tree size defined as the number $|\widetilde{T}|$ of leaves of the tree T:

$$R_\alpha(T) = R(T) + \alpha|\widetilde{T}|, \tag{2.79}$$

where $R(T)$ is a standard misclassification cost.

To determine the optimal value of α, Breiman et al. (1984) defined validation procedures which estimate performance of the candidate α values and select the best one. They have proven some important properties of the formula, which revealed that the pursuit of the optimal value of α can be efficient. First of all, they noticed the following property:

Property 2.1 For each value of α there is a unique smallest subtree $T_\alpha \prec T$ minimizing R_α.

It means that if any other subtree $T' \prec T$ also minimizes R_α, then $T_\alpha \prec T'$.

Further reasoning proved a theorem rephrased as the following property, which laid foundation for Cost-complexity optimizationcost-complexity validation methodology.

Property 2.2 There exist: a unique increasing sequence $\alpha_1, \ldots, \alpha_n$ and a decreasing sequence of trees $T_1 \succ \cdots \succ T_n$, such that $\alpha_1 = 0$, $|\widetilde{T_n}| = 1$ and for all $i = 1, \ldots, n$, T_i minimizes R_α for each $\alpha \in [\alpha_i, \alpha_{i+1})$ (to be precise, we need to define additional $\alpha_{n+1} = \infty$).

Less formally: there exists a unique decreasing sequence of αs that explores all possible trees optimizing R_α, and the trees are also precisely ordered. A natural result is that to determine all the αs, one can act sequentially, starting with the whole tree and determining nodes to be pruned, one by one (when it happens that pruning two or more nodes is conditioned by the same value of α, they must be pruned together). Formally, algorithm 2.12 presents the method. It is important from the point of view of algorithm complexity, that after pruning a node only the nodes on the path to the root need an update of their αs—recalculating all of them would be a serious workload.

Provided the algorithm to determine the sequence of αs and corresponding smallest trees, the optimal α and the optimally pruned tree can be selected, for example, according to Algorithm 2.13.

Algorithm 2.12 (Determining αs and their optimal trees for cost-complexity minimization)

***Prototype:* CCAlphasAndTrees(D,Learner)**

Input: *Training data D, DT induction method Learner.*

Output: *Sequences of αs and trees as in property 2.2.*

The algorithm:

```
1. $T \leftarrow Learner.LearnFrom(D)$—induce full DT to be analyzed
2. Determine threshold $\alpha_N$ for each node $N$ of $T$
3. $i \leftarrow 1$
4. $\alpha \leftarrow 0$
5. repeat
     a. Prune all nodes $N$ of $T$ with $\alpha_N = \alpha$ (modifies $T$)
     b. Update all $\alpha_N$ for nodes $N$ on the path from the root to just pruned node(s)
     c. $T_i \leftarrow T$
     d. $\alpha_i \leftarrow \alpha$
     e. $\alpha \leftarrow \min_{N \in T} \alpha_N$
     f. $i \leftarrow i + 1$
   until $|\widetilde{T}| = 1$
6. return $\alpha_1, \ldots, \alpha_{i-1}$ and $T_1, \ldots, T_{i-1}$
```

The method assumes that two separate datasets are input: one for tree construction and one for validation. After a tree is built, its all sensible (according to cost-complexity minimization rule) pruned trees are determined and their accuracy estimated by classification of the validation data. The smallest tree with minimum classification error is selected as the validated pruned tree.

The situation is slightly more complex, when multiple validation is performed (for example Cross-validation). In each pass of the CV, a DT is built and analyzed to determine all the threshold αs and their smallest trees. Similarly, in the final pass, a DT is trained on the whole dataset and the two sequences are determined. Unfortunately, the series of αs in each pass of CV and in the final pass may be all different. To select a winner α all values of the final pass must be examined and average accuracy estimated for them. Since the sequence contains threshold αs, it is not the most sensible to check how they would behave in CV. In place of the border values α_i, Breiman et al. (1984) proposed using geometrical averages

$$\alpha'_i = \sqrt{\alpha_i \cdot \alpha_{i+1}}. \tag{2.80}$$

Algorithm 2.13 (Cost-complexity minimization with external validation data)

Prototype: ***CCTrnVal(***D_{trn}**,**D_{val}**,Learner)**
Input: *Training data* D_{trn}*, validation data* D_{val}*, DT induction method Learner.*
Output: *Optimally pruned DT, optimal* α*.*
The algorithm:

```
1. $((\alpha_i), (T_i)) \leftarrow CCAlphasAndTrees(D_{trn}, Learner)$
2. for $i = 1, \ldots, n$ do
       $R_{val}(\alpha_i) \leftarrow$ misclassification error of $T_i$ measured for $D_{val}$
3. $opt \leftarrow \arg\min_{i=1,\ldots,n} R_{val}(\alpha_i)$
4. return $T_{opt}$ and $\alpha_{opt}$
```

For each α'_i proper validation errors are extracted from CV and averaged. The largest α' with minimum average is the winner. Formal notation of the algorithm is presented as Algorithm 2.14. In line 4 of the algorithm, it should be specified what is meant by α_{j+1} for maximum j. Breiman et al. (1984) do not propose a solution, but since it is the matter of the largest α in the sequence, it is reasonable to assume $\alpha_{j+1} = \infty$, as further increasing α to infinity does not change the corresponding optimal tree—for all values of α in the range $[\alpha_j, \infty)$, the optimal tree is a stub with no split, classifying to the class with maximum prior probability. Such definition is advantageous, because in all the series, always the last $\alpha = \infty$ corresponds to maximally pruned tree, where the root acts as a leaf (sort of baseline, majority classifier).

Algorithm 2.14 (Cost-complexity minimization with CV-based validation)

Prototype: ***CCCV(***D**,Learner,**n**)**
Input: *Training data D, DT induction method Learner, number of CV folds n.*
Output: *Optimally pruned DT, optimal* α*.*
The algorithm:

```
1. Prepare training-validation data splits: $(D^t_1, D^v_1), \ldots, (D^t_n, D^v_n)$
2. for $i = 1, \ldots, n$ do
       $\left((\alpha^i_j), (T^i_j)\right) \leftarrow CCAlphasAndTrees(D^t_i, Learner)$
3. $\left((\alpha_j), (T_j)\right) \leftarrow CCAlphasAndTrees(D, Learner)$
4. for each $\alpha_j$ do
       $R_{CV}(\alpha_j) \leftarrow \frac{1}{n}\sum_{i=1}^{n} R_{val}\left(\sqrt{\alpha_j \cdot \alpha_{j+1}}\right)$
5. $opt \leftarrow \arg\min_j R_{CV}(\alpha_j)$
6. return $T_{opt}$ and $\alpha_{opt}$
```

2.4.3.3 Degree-Based Tree Validation

The idea of the *degree of pruning* applied in SSV DT (Grąbczewski and Duch 1999, 2000) is based on counting differences between reclassification errors of a node and its descendants. Pruning with given degree (which is an integer) means pruning the splits, for which the difference is not greater than the degree. The rationale behind the definition is that the degree defines the level of details in DT and that for decision trees trained on similar data (in CV, large parts of training data are the same), optimal pruning should require similar level of details. The level of details can be described by the number of leaves in the tree, but then, an additional method is needed for deciding which nodes should be pruned and which should be left. The definition of degree of pruning clearly determines the order in which nodes are pruned. Properties analogous to 1 and 2 are in this case trivial, so are not reformulated here. To analyze all possible degrees of pruning, tree nodes are pruned one by one in the order of increasing differences between node reclassification error and the sum of errors of node children. Such value is immutable for each node, so once determined it does not need recalculation. Therefore the algorithm collecting pairs of degrees and optimal trees is slightly simpler than the corresponding algorithm for Cost-complexity optimizationcost-complexity minimization.

Algorithm 2.15 (Determining degrees of pruning and trees pruned to degrees)

***Prototype:* DegreesAndTrees(D,Learner)**
Input: *Training data D, DT induction method Learner.*
Output: *Sequences of degrees and pruned trees.*
The algorithm:

```
1. $T \leftarrow Learner.LearnFrom(D)$—induce full DT to be analyzed
2. Determine the degrees of pruning $d_N$ for each node $N$ of $T$
3. $i \leftarrow 1$
4. for each degree $d$ determined above, in increasing order do
       a. for each node $N$ of $T$, with all subnodes being leaves do
              if $d_N <= d$ then prune node $N$ (change into a leaf)
       b. $d_i \leftarrow d$
       c. $T_i \leftarrow T$
       d. $i \leftarrow i + 1$
5. return $(d_1, \ldots, d_{i-1})$ and $(T_1, \ldots, T_{i-1})$
```

Also the main algorithms performing validation and selecting the best pruned trees get simpler. To save space only the one based on CV is presented (Algorithm 2.16). In the case of degrees, no geometrical averages are applied. Average risk is calculated directly from CV tests, where the errors for particular pruning degrees can be easily read out.

Instead of direct optimization of the pruning degree, one can optimize tree size (defined as the number of leaves) and use the concept of pruning degrees, just to

Algorithm 2.16 (Degree-based DT validation based on CV)

Prototype: ***DegCV****(D,****Learner****,n)*
Input: *Training data D, DT induction method Learner, number of CV folds n.*
Output: *Optimally pruned DT, optimal degree d.*
The algorithm:

```
1. Prepare training-validation data splits: $(D_1^t, D_1^v)$, ..., $(D_n^t, D_n^v)$
2. for $i = 1, \dots, n$ do
     $\left((d_j^i), (T_j^i)\right) \leftarrow DegreesAndTrees(D_i^t, Learner)$
3. $((d_j), (T_j)) \leftarrow DegreesAndTrees(D, Learner)$
4. For each $d_j$:
     $R_{CV}(d_j) \leftarrow \frac{1}{n}\sum_{i=1}^{n} R_{val}(d_j)$
5. $opt \leftarrow \arg\min_j R_{CV}(d_j)$
6. return $T_{opt}$ and $d_{opt}$
```

determine the order in which the nodes are pruned. To obtain such methods, it suffices to modify Algorithms 2.15 and 2.16 to handle sequences of leaves counts in place of the sequences of degrees.

This method is much simpler, easier to implement and a bit faster than the cost-complexity optimization of Breiman et al. (1984).

2.4.3.4 Optimal DT Pruning

Bohanec and Bratko (1994) proposed the OPT algorithm for construction of the *optimal pruning sequence* for given DT. By means of Dynamic programming they determined an optimal subtree (maximizing training data reclassification accuracy) for each potential tree size. Then, depending on which final accuracy they were interested in, they pruned the tree to the appropriate size. Algorithm 2.17 presents the main procedure of the method. It generates sequences of error counts (for the training data) and pruned nodes collections for integer arguments denoting the decrease in the number of leaves, corresponding to subsequent sequence positions. The main procedure recursively gets the sequences for each child of the node being examined, and combines the results with the *Combine()* method presented as Algorithm 2.18, in a way compatible with the paradigm of Dynamic programming.

To be precise, there is a little difference between this formulation of the OPT algorithm and the original one: Bohanec and Bratko (1994) did not record the error counts in one of the result sequences (here named **E**), but the difference in error count between the node acting as a leaf and the whole subtree rooted at the node. The difference is null when the full tree is 100 % accurate. It does not matter either, when only one tree is analyzed. Some differences in decisions can appear when multiple validation is performed (for example Cross-validation) to average the scores and draw

Algorithm 2.17 (OPTimal DT pruning sequence)

***Prototype:* *OPT*(N)**

Input: *Tree node N (let $n = |\widetilde{T^N}|$).*

Output: *Sequences of reclassification errors $\mathbf{E} = (E[i])$ and collections of nodes to be pruned $\mathbf{P} = (P[i])$ (for each sensible leaves reduction $i = 0, \ldots, n-1$).*

The algorithm:

```
1. $P[0] \leftarrow \emptyset$
2. $E[0] \leftarrow 0$
3. for $i = 1, \ldots, n-1$ do
     $E[i] \leftarrow -1$
4. for each subnode $M$ of $N$ do
     a. $(\mathbf{E}_M, \mathbf{P}_M) \leftarrow OPT(M)$
     b. $(\mathbf{E}, \mathbf{P}) \leftarrow Combine(\mathbf{E}, \mathbf{P}, \mathbf{E}_M, \mathbf{P}_M)$
5. if $N$ has subnodes then
     $P[n-1] \leftarrow \{N\}$
6. $E[n-1] \leftarrow e_N$ /* $e_N$ is the number of errors made by $N$ for the training data */
7. return $E$ and $P$
```

Algorithm 2.18 (Combining OPT sequences)

***Prototype:* *Combine*($\mathbf{E}_1, \mathbf{P}_1, \mathbf{E}_2, \mathbf{P}_2$)**

Input: *Sequences of error counts and collections of nodes to be pruned ($\mathbf{E}_1, \mathbf{P}_1, \mathbf{E}_2, \mathbf{P}_2$, indexed by $0, \ldots, n_1$ and $0, \ldots, n_2$ respectively).*

Output: *Combined sequence of reclassification errors $\mathbf{E}$ and collections of nodes to be pruned $\mathbf{P}$.*

The algorithm:

```
1. $P[0] \leftarrow \emptyset$
2. $E[0] \leftarrow 0$
3. for $i = 1, \ldots, n-1$ do
     $E[i] \leftarrow -1$
4. for $i = 0, \ldots, n_1$ do if $E_1[i] \geq 0$ then
     for $i = 0, \ldots, n_2$ do if $E_2[i] \geq 0$ then
       a. $k \leftarrow i + j$
       b. $e \leftarrow E_1[i] + E_2[j]$
       c. if $E[k] < 0$ or $e < E[k]$ then
            i. $P[k] \leftarrow P_1[i] \cup P_2[j]$
            ii. $E[k] = e$
5. return $E$ and $P$
```

some conclusions from the means, but the probability of differences in decisions is not high.

For binary trees, the sequences are full (no -1 value is left in the **E** sequence, because it is possible to get any tree size with pruning. When the pruned tree contains multi-way splits, it may be impossible to get some sizes, so in the final **E** sequence, some elements may stay equal to -1 after the optimization.

The original algorithm was justified from another point of view than improving classification of unseen data (generalization). Instead, the authors assumed that the full tree was maximally accurate and searched for the smallest tree preserving given level of accuracy. Although the motivation for DT validation is different, the same Dynamic programming scheme can be used to determine expected accuracy of trees of each size, inside a Cross-validation. The size maximizing expected accuracy becomes the size of the target tree obtained with optimal pruning of the tree generated for the whole training data.

The idea is quite similar to that of the degree-based pruning, however it is much more detailed, in the sense that it analyzes all possible tree sizes, while in the former algorithm only those resulting from pruning to a given degree (two subsequent trees may have sizes differing by quite large number, because increasing the pruning degree from 1 to 2 may prune many nodes in the tree). Naturally, the cost paid for the increased level of details is an increase of computational complexity of the method—the dynamic programming optimizations cost much more, so in the case of large trees the time of learning may get significantly larger. Almuallim (1996) offered some improvements in the calculations, which under some assumptions reduces the computational costs, but they are not always helpful. They can be applied when we are interested in a particular tree size, but for example, in the tests described in Chap. 5 where many different tree sizes are examined, the simplifications proposed by Almuallim (1996) are not applicable.

2.5 Search Methods for Decision Tree Induction

Search methods and tree structures are inextricably linked with each other. Traces of search procedures have the form of trees, and inversely: tree construction is naturally obtained with search methods. Therefore, also in the area of decision trees, search methods serve as fundamental tools of induction. The simplest and usually the fastest search technique is the greedy one, which at each stage preforms a local minimization to determine the next state and never returns to the previous stages in order to try alternative solutions. Apart from the term *greedy search* such technique can be called many other names. Because of no returns and the goal of class separation (each split may improve the separation and never deteriorate it), the techniques of *depth first search*, *best first search*, *hill climbing* and some others are all equivalent in the sense that they end up with the same tree (just the order of nodes creation may be different, but it does not cause differences in the final models). As a result, they all can be seen as the top-down greedy DT induction method formalized as Algorithm 2.1. Such technique is used in vast majority of DT induction approaches, but is not the only one applicable to the problem. Some applications of Beam search and Lookahead search, described below, have also been examined.

General Search Aspects

Regardless of the search method used for DT induction it is always important to avoid unnecessary calculations. When a node split is regarded, a set of split candidates must be analyzed and the best one selected. The list of candidates must be determined reasonably, because many of the seeming candidates can be judged in advance as certainly worse than some others and ignored with no change to the final tree.

When the optimal (with respect to a given criterion) binary split of a continuous feature is to be found, it is natural that the sensible split points are only those lying between the values observed in the training data. All split points lying between the same two adjacent values observed in the training data bring the same split result, so in practice, only the points in the middle of the interval are taken into account. Moreover, split quality measures usually have a form of Concave functions or have other properties that justify ignoring the split points lying between objects belonging to the same class, because separating them gives lower scores than putting the neighbors together to one or the other side of the split. Proofs of such properties, with special attention paid to particular methods, have been published for example by Breiman et al. (1984, Gini index), Fayyad and Irani (1992b, Information gain) Grąbczewski (2003, SSV criterion).

In the case of unordered features, the splits are not determined by a point (a real value) but by division of the sets of symbols into disjoint and complementary subsets. A subnode is created for each subset, and the training data objects are distributed to the subnodes, according to the symbols describing them. Some algorithms (for example C4.5) consider only singletons and split into as many subnodes as the number of possible symbols of the split feature. In such approach there is just one way to split on the basis of a symbolic feature. The most serious drawbacks of such solutions are that they can split the training data into many small datasets without significant reason, and that split quality measures may give overoptimistic estimates of the symbolic splits, because of accidental correlation between the (numerous) symbols and assigned classes. Therefore, often, binary splits are preferred also for symbolic features. But the number of possible splits may get huge if the number of possible symbols is large. To check all possible splits one needs to examine $2^{s-1} - 1$ candidates, where s is the number of possible symbols of the feature. The number comes from the count of all subsets of an s-element set (2^s) and the fact that neither the empty set nor full set of symbols can be accepted to determine a subnode, and that a subset and its complement determine the same split, so only one of them should be used: $(2^s - 2)/2 = 2^{s-1} - 1$. When the number s is large, analyzing each possible split may be unfeasible and additional restrictions must be applied to reduce the amount of calculations. A solution based on a subset generator was described in Sect. 2.2.7 on the SSV algorithm.

Another aspect of splitting on the basis of single features is bias in feature selection. When two features are equally informative, they should have the same probability of being selected as the split feature. If the number of analyzed splits is significantly different for ordered and unordered features, then a bias can be observed in favor of one of the types. More thorough discussion of the bias in feature selection for DT induction is presented in Sect. 2.7.

Lookahead Search

One of the possibilities of more thorough search is to use the methods called *lookahead search*. As suggested by the name, such algorithms grow subsequent DT branches not on the basis of direct split quality measurements, but with some forward insight to the potential gains of using particular splits. After hypothetical acceptance of a split, the resulting subnodes are further split, in order to check the quality of the best possible depth-restricted subtree rooted at the node (the depth is given by a parameter). The search with depth n is also called an *n-ply lookahead search*. The algorithms resemble mini-max search used in game playing, but here, the decisions at subsequent levels are made by cooperating parties, not by adversaries (with the same, not competitive, quality measures).

Algorithm 2.19 (Lookahead split selection)

Prototype: LookaheadBestSplit(depth,D)
Input: *Lookahead depth, training dataset D.*
Output: *The split estimated as the best.*
The algorithm:

```
1. for each $s \in CandidateSplits(D)$ do
      $N_s \leftarrow SplitAhead(depth, D, s)$
2. $best \leftarrow \arg\max_s Quality(T^{N_s})$
3. return $best$
```

Formally, we can see the lookahead search algorithms as ordinary top-down DT induction methods (Algorithm 2.1) with the procedure of best split selection based on some forward insight. So in fact, it is not a new search method, but a special split selection method, although its intuitive perception may be different. A natural instance of such lookahead split selection is presented as Algorithm 2.19. It builds a branch of a pre-defined depth for each candidate split (with the *SplitAhead()* method presented as Algorithm 2.20) and estimates the quality of the split hierarchy. The split providing the highest quality of the generated branch is returned as the result of the lookahead split selection. It is important to realize that the quality measure used in such procedure must be ready for assessment of tree structures, not just single splits. However, each Split quality measure designed to estimate multipart splits is

Algorithm 2.20 (Depth-limited splits)

Prototype: SplitAhead(depth, D, s)
Input: *Lookahead depth, training dataset D, initial split s.*
Output: *The root node of the created tree.*
The algorithm:

```
1. if $s \neq \perp$ and $depth > 0$ then
     a. $\{D_1, \ldots, D_n\} \leftarrow s(D)$ /* split the node data */
     b. for $i = 1, \ldots, n$ do
          i. $best \leftarrow BestSplit(D_i)$
         ii. $N_i \leftarrow SplitAhead(depth - 1, D_i, best)$
     c. $Children \leftarrow (N_1, \ldots, N_n)$
   else
     $Children \leftarrow \perp$
2. return $(D, s, Children)$
```

naturally applicable, because each tree can be treated as a single split into the parts corresponding to its leaves.

The lookahead split selection can be called a meta-level split procedure, as it uses external method of node split selection (*BestSplit()*) to build small trees and estimate their quality. Also the methods *CandidateSplits()* and *Quality()* can be arbitrarily chosen to obtain different effects. In fact, the three functions mentioned above are the parameters of *LookaheadBestSplit()*, but are not listed explicitly in the parameter list to keep the code clearer.

The number of splits to be made in a lookahead estimation, grows exponentially with the *depth* parameter, so to avoid large computational overhead, it is not recommended to look deeper than just one level (*depth=1*).

Beam Search

Greedy selection of the best split at each node would be the only sensible technique, if the split quality estimation were perfect. As discussed in the beginning of Chap. 2, optimization of a quality measure at a single node is not the same as optimization of the overall tree, so sometimes it may be more adequate to select a split of lower local quality, but providing better data environment for further branch splits, and in effect, bringing shorter or more accurate trees.

A tool facilitating a number of top-ranked partial solutions to take part in further pursuit of maximum overall quality is the beam search—the process conducted in almost the same way as the breadth-first search, but with a limit on the number of states that can be explored at each level, to prevent combinatorial explosion. In decision tree induction, the beam is a container for a number of top-ranked partial trees, which are developed in parallel. So the main difference from the standard greedy approach is that at each stage of the search, the focus is not only on a single tree but on all models contained within the beam. The size of the beam container (*beam width*) is a parameter of the method.

In general, the algorithm operates on so called states (here a state is a tree developed so far) and iteratively generates new states from the current ones, with the restriction that only the best ones are placed in the beam and further explored. Algorithm 2.21 presents the scheme formally. It is a general, abstract procedure capable of handling any kind of states, not just decision trees. At the beginning there is a single initial state in the beam. In each iteration, all possible children states of the trees contained in the beam are generated, and the best w of them (the beam width parameter) are placed in the beam. Since the goal is a single final tree, the search is stopped when a final state (a complete tree) is found.

Algorithm 2.21 (Beam search)

```
Prototype: BeamSearch($S$, $w$)
Input: Initial search state $S$, beam width $w$.
Output: Final search state.
The algorithm:
    1. beam $\leftarrow \{S\}$
    2. while NoFinalState(beam) do
        a. children $\leftarrow \emptyset$
        b. for each state $\in$ beam do
               children $\leftarrow$ children $\cup$ ChildrenStates(state)
        c. beam $\leftarrow$ BestStates($w$, children)
    3. return the final state in beam
```

Apart from the explicit parameters of the code, the functions called within the main procedure also significantly influence the process and its results. The most important of the subroutines is *BestStates()* which selects the best trees to be put into the beam of width w. It may compare the children states (here the trees) according to many different criteria like model accuracy, purity of decision tree leaves, information measures, MDL criteria and so on. The fundamental difference between application of these measures and of those used for single split quality assessment is that they need to compare the whole trees, not single splits. In particular, similarly to the measures used in the lookahead approach, they have to be applicable to multi-way splits, because nontrivial trees may split the space into large number of parts. The other subroutines (*NoFinalState()* and *ChildrenStates()*) can also be implemented in various ways, but their goals are rather straightforward, when the aim of the main search strategy is precisely defined.

The practice of filling the beam with the best states shows that trivial selection of the models maximizing criteria like accuracy, amount of information and others is not the most successful choice. When many children states are generated for each state in the beam, it often happens that after two or three iterations, all trees in the beam are very similar. For example, after the first iteration, the beam contains w single split trees with different features used for the split, but after the second iteration, the beam contains w children of the same single split tree. As a result, the time consumption is significantly larger, but the time is wasted for exploration of almost the same areas of the model space. A remedy for this is nontrivial beam selection equipped with tools protecting from appropriation of the whole beam by a close family of trees. The tools may be some measures of diversity, for example, the ones proposed in the field of data clustering (Hubert and Arabie 1985; Vinh et al. 2010), as the feature space partition given by a tree ideally fits the assumption of clustering and similarity between trees may be determined by means of similarity between the space partitions they determine.

Broader search means larger complexity of the procedure, so in comparison to simple top-down greedy DT induction, beam search is naturally more expensive. The complexity grows by the factor of w (beam width), because in each iteration, w trees are examined in place of a single one of the greedy approach. Obviously, one can give examples, where beam search ends in a shorter time than the greedy search and inversely, where the relative cost factor of beam search is much greater than w. The former situation may occur when the most attractive split at the root of the tree requires complicated splits in subsequent levels, while some less attractive single split perfectly matches some other splits and yields small and accurate tree. The opposite relation can be observed when beam search rejects the direct solution of the greedy search, because of finding other seemingly more attractive solutions, which turn out to be a blind alley.

Beam search has been found attractive not only in searching for single DTs. Grąbczewski and Duch (2002a, 2002b) have found it useful for generation of a number of trees to act in ensembles (Decision forests). Small modifications of Algorithm 2.21 to stop the search after a specified number of final states (not just the first one) is found, gives an opportunity to build forests without much additional computational effort.

Restrictions on the beam width can be helpful, when combined with the lookahead search described above (Buntine 1993). It can reduce the increase of computation time requirements when the lookahead depth is greater than 1.

The Danger of Oversearching

Some authors have reported that using more thorough search procedures (than the most common, simple hill climbing) is often assisted by a decline in models accuracy (Murthy and Salzberg 1995; Quinlan and Cameron-Jones 1995; Segal 1996; Janssen and Fürnkranz 2009). Their observations mostly concerned classification rules induction, not decision tree algorithms, however the tasks are so similar, that they should be subject to the same effects, if the conclusions are derived from general, reliably prepared experiments.

Murthy and Salzberg (1995) compared greedy DT induction algorithm with a lookahead approach and concluded that both methods are similarly accurate on average and pointed out the pathology, that the lookahead approach can generate larger and less accurate trees than the greedy strategy. On average, the trees obtained with lookahead turned out to be shallower, although in many cases it was just the matter of balance, lacking in the results of the greedy search. The authors proposed a procedure of balancing the trees based on rotations (without changes to the decision function). Another conclusion was that pruning can provide better results than lookahead, because the trees pruned with the cost-complexity optimizationcost-complexity minimization method (see Sect. 2.4.3.2) to prune the trees on the basis of a validation dataset consisting of 10 % of the training data excluded before tree construction. The observation is reasonable, but should not be treated as an argument against more thorough search procedures, as search and pruning should be perceived rather as two complementary, not alternative techniques. It seems that larger test errors of the lookahead method were not a consequence of oversearching, but the effect of pruning the trees coming from the competing algorithm.

Quinlan and Cameron-Jones (1995) performed some test of beam search applied to a classification-rules-induction method, where model generalization was controlled by Laplace error correction (see Eq. (2.71). They presented some experiments where the rules minimizing the Laplace error estimates turned out to be the less accurate the larger beam width was used (in the range 1–512 with exponential growth). As a proposed solution they introduced the *layered search* method which can be seen as beam search with changing beam width. They started the search with beam width equal to 1 (corresponding to hill climbing) and then, in each iteration doubled it. Such strategy gave more attractive results than both hill climbing and beam search.

The experiments of Quinlan and Cameron-Jones (1995) were repeated and analyzed by Segal (1996), who noticed that the Laplace error estimation of single rule is not much correlated with the true error rate calculated on external test dataset. The "oversearching effect" was a result of poor match between the evaluation function used for training and the test performance measure. Low accuracies were the result of ordinary overfitting. Segal (1996) noticed that Laplace error estimation trades off accuracy for coverage, which is detected by accuracy tests on external data. They suggested a modification of the Laplace correction, named *LaplaceDepth*, calculated for each rule R, and defined as

$$LaplaceDepth(R) = 1 - \frac{n_c + d \cdot p_a(c)}{n + d} \tag{2.81}$$

where c is the decision class of the rule R, $p_a(c)$ is the prior probability of class c, n is the number of examples covered by R, n_c is the number of objects from class c satisfying the rule and d is a parameter. Such evaluation criterion is compatible with the m-probability-estimation (see Eq. (2.72)). Here the m parameter is named d and, in the proposed strategy, should be equal to the length of the rule. The modification of the rule evaluation criterion practically removed the effect of "oversearching". Segal (1996) suggested that other measures of rule quality may do even better than the improved Laplace error.

More recently, Janssen and Fürnkranz (2008, 2009) revisited the problem and, also seeking the causes of the conclusions about oversearching in Laplace error estimation, examined nine different heuristics (including Laplace corrected error) in experiments with three search strategies applied to the task of decision rule induction. The search methods they examined were: hill climbing, beam search and ordered exhaustive search. The latter is an implementation of exhaustive search, optimized to avoid repeated generation of the same rules. Significant simplification was possible because of the specificity of the definition of the rule induction problem, where the form of rules was limited, for example, in such a way that two premises could not use the same feature, so the features could be analyzed in a strictly defined order. From illustrations in the article, one can infer that the beam search also had a specific form. One of the figures and some text suggest that each new beam is created in a way supporting one child of each state from the previous beam. This is compatible with the heuristics aimed at beam diversity, described above in the description of the beam search approaches.

The experiments of Janssen and Fürnkranz (2009) also show that when the error estimation with Laplace correction is used, the test accuracy tends to deteriorate with increasing beam width. Nevertheless, for other measures, the results are completely different. Evaluating the rules by their precision gives stable accuracy plots, almost constant in the whole range of beam width, while the rules get simpler with larger beam widths. An example utterly opposite to Laplace error is the *odds ratio* for which a significant increase of accuracy was observed with increasing beam width.

To sum up, the conclusions from all the experiments performed to research the aspects of oversearching in DT and rule induction are not consistent. Sometimes, more exhaustive search improves the results and sometimes deteriorates them. Dependence on rule quality measures has been reported, but the notifications require more evidence. The only certain conclusion is that more advanced search procedures should be used with special care, but should not be forgotten. Future meta-learning algorithms will certainly help make decisions also about search method selection.

2.6 Decision Making with Tree Structures

Regardless of the strategy of decision tree induction, a method for final decision making must be selected. The most common approaches are based on the distribution of data objects of different classes falling into proper DT leaves. When a new data object is to be classified with a DT, it traverses the tree to discover the most adequate leaf. Most often, the object is assigned the label of the class dominating within the leaf. If the classification decision is to be presented in the form of probabilities of belonging to particular classes, they are usually estimated on the basis of that leaf (more precisely the training data falling into the leaf), and sometimes on the basis of all the nodes on the path to the leaf.

Probabilities as Proportions

The simplest method to obtain probability estimates is to calculate the proportions of objects within the leaf belonging to particular class and all the objects in the leaf:

$$P(c|x) = \frac{n_{L(x),c}}{n_{L(x)}}, \tag{2.82}$$

where $L(x)$ is the leaf adequate for x. The same notation is used in subsequent formulae.

Laplace Correction

When the counts of objects within the leaf are small, the proportions do not estimate real probabilities accurately. For example, when just three or four objects (all of the same class) fall into a leaf, proportion-based probabilities are binary, claiming that the probability of finding a representative of any other class in the area is zero. Usually, such extreme claims are not true. Therefore, some corrections have been proposed and successfully used in many applications. One of the methods, called *Laplace correction* (or *Laplace's law of succession*), has already been introduced in the context of Minimum Error Pruning (see Sect. 2.4.2.3) and is calculated as:

$$P(c|x) = \frac{n_{L(x),c} + 1}{n_{L(x)} + k}, \tag{2.83}$$

where k in the number of classes in the classification problem.

m-Probability-Estimation

Another way of probability correction was proposed by Cestnik and Bratko (1991) and called *m-probability-estimation*. It has also been introduced in the section on Minimum Error Pruning. The corrected probabilities, referred to as *m-estimates*, are defined by:

$$P(c|x) = \frac{n_{L(x),c} + m \cdot p_a(c)}{n_{L(x)} + m}, \tag{2.84}$$

where $p_a(c)$ is the a priori probability of class c and m is the method parameter. When the a priori probabilities of all the classes are equal, by setting m to the number of classes, we obtain the formula equivalent to the Laplace correction. In general, the parameter m defines the "strength of pushing" the proportions towards a priori probabilities. In a couple of applications, the authors of the idea proposed using $m = 2$. It is important to realize that this method may result in different winner class than the one resulting from pure proportions or those with Laplace correction—when $m \to \infty$, the probabilities converge to the priors, so the winning class may be different than the one dominating the data sample of the leaf. Laplace version of the estimation is compatible with pure proportions when just the winner is to be determined, but of course, the probabilities have different values, so for example, combined with others in an ensemble, may also bring significantly different results.

Path Combined Majority

In LTree family of algorithms Gama (Gama 1997), it is proposed that the decision making strategy respects class distributions in the nodes of the whole path of the tree, followed by the data item to be classified. Gama (1997) suggested to calculate probabilities for each tree node, starting with the root node, down to the leaves, in such a way that only the root node probabilities are estimated as proportions and for each non-root node N, its proportions and the probabilities calculated for the parent node contribute to the final probability estimates:

$$P(c|N) = \frac{P(c|Parent(N)) + w\frac{n_c}{n_N}}{1+w}, \tag{2.85}$$

where w is a method parameter. With $w = 1$ (suggested by the author), if the path from the root node to the leaf $N_L = L(x)$ is $N_0, \dots, N_L$, then the probability

$$P(c|x) = P(c|N_L) = \sum_{i=0}^{L} \frac{1}{2^{L-i}} \frac{n_{N_i,c}}{n_{N_i}}. \tag{2.86}$$

Similarly to the m-probability-estimation approach, including class proportions of parent nodes may significantly change the decisions in relation to the ones calculated on the basis of the leaf only. Unlike the m-estimates, in this approach, the probabilities are not pushed towards the prior probabilities, but towards probability distributions in larger groups of objects (parent nodes). Although the contribution of the root node plays similar role as m-estimates, because the proportions in the root (that is, in the whole training data sample) are exactly the same as the estimates of priors in m-probability-estimation, the influence of the root node on the final values is very small for longer paths, as its factor is $\frac{1}{2^L}$.

More sophisticated combinations of path nodes proportions were proposed by Buntine (1993) and Kohavi and Kunz (1997) as part of their Option Decision Trees approach. In a Bayesian analysis, they define weights for ensembles of classifiers extracted from trees. The weights can combine decisions, not only of the options (different trees), but also of the nodes of a single path. Indeed, the nodes on a path from the root to a leaf can be treated as separate classifiers combined into an ensemble. Some more information about the approach can be found in Sect. 2.8.1.

2.7 Unbiased Feature Selection

One of the aspects of decision trees interpretation is feature relevance. When splits are performed on the basis of single features, it is reasonable to expect that the feature occurring in a tree node is, in some sense, the most informative one in the context of class discrimination among the data sample of the node. However, the term “the most informative” is not precise, so there is not a unique measure of such feature relevance. Otherwise, there would exist a single, the most appropriate way to split DT nodes, and one method would be sufficient for all purposes. Therefore, so many different approaches to DT induction have been undertaken, and still none has been announced to outperform all the others.

Despite the fact, that no universal measure exists to estimate feature relevance, many researchers have been struggling for solutions of the problem of *unbiased feature selection* for DT splits. The goal of their research has been to provide methods of DT induction, that would not favor features because of their accidental relationships with the target variable.

So fair selection of split variable would provide very valuable information about the most discriminative features of the data table at hand in the context of particular task. DT techniques have been successfully used to extract information about feature importance for the purpose of feature ranking and selection (Duch et al. 2002, 2003; Grąbczewski 2004; Grąbczewski and Jankowski 2005), but still, conclusions about feature significance on the basis of feature selection made by a DT induction algorithm should be drawn with special care, especially when strong interaction between features can be observed.

The problem of variable selection bias has been observed from the early stages of research on DT induction. In AID (see Sect. 2.9 and Morgan and Sonquist 1963a,b), performing binary splits by means of dividing the set of possible feature symbols into two disjoint subsets, more categories of a feature means more split possibilities, resulting in a bias in favor of the features with many symbols. Kass (1980) proposed CHAID as a modification of the method to perform significance testing, so as to nullify the bias.

Similarly, in the efforts of DT induction based on miscellaneous heuristics instead of statistical tests, such as the one of the pioneer method, ID3, symbolic features with many possible values are favored, because the more symbols, the larger probability of coincidental correlation between the feature and the target variable. An extreme case is a feature with so many values that no value is repeated in the training data. Then, a perfect split can always be done, but it can be fully accidental, providing no sensible information, so that a tree using the split is very unlikely to generalize well the knowledge behind the training data. Probably the most popular enterprise to reduce the bias was introduction of information gain ratio in place of information gain in C4.5 algorithm (see Sect. 2.2.3 and Quinlan 1993).

The problem of bias is not specific only to selecting among unordered features with various counts of possible symbols. It also concerns comparisons between symbolic and numeric variables. Moreover, in different methods the bias may be in favor of different groups of features, for example, some methods are biased in favor of multi-valued symbolic features and some others against them.

Statistical Methods

One of the most popular idea in the approaches to unbiased variable selection for DT splits is separation of the procedures of feature selection and final split definition (feature selection and split-point selection).

In so called statistical trees, it has been often realized by application of statistical tests to estimate feature importance as an opposite of independence between the feature and the target variable.

A turn to such statistical approach was made by White and Liu (1994), who proposed using χ^2 distribution for the purpose of attribute selection instead of measures like information gain, information gain ratio and Mántaras distance (de Mántaras 1991), which are biased in favor of attributes with large numbers of symbols. The approaches utilizing χ^2 distribution are preferable, because they facilitate sensible comparisons of results calculated for different attributes. Compensation of the dependence on the number of cells in the contingency tables being analyzed, was obtained by using χ^2 probability instead of the test statistic, as the probabilities are comparable also in the case of different distribution parameters (degrees of freedom), while the values of test statistics are not. White and Liu (1994) also proposed using G statistic, defined in the language of information theory for the same purpose.

The idea of using statistical test p-values instead of test statistics directly has been applied by many more authors in their approaches to unbiased comparisons. In fact, it is the most commonly used tool of bias elimination efforts.

White and Liu (1994) illustrated their conclusions with simulation results on datasets with three different class distributions, but all with the assumption of independence between the class variable and the predictors.

Because the probabilities coming from the χ^2 and G statistics can be approximated with large error when the expected frequencies are small, White and Liu (1994) suggested using Fisher's exact probability test (see Appendix section A.2.4) instead, in the case of two-class problems. For multi-class tasks, similar approaches can be developed.

Both ideas of separating feature selection from split selection and making decisions on the basis of p-values of statistical tests have been extensively explored and applied to several interesting DT induction algorithms proposed by the group of prof. Loh: FACT, QUEST, CRUISE, GUIDE and others (see Sect. 2.2.5 and Loh and Vanichsetakul 1988; Loh and Shih 1997; Kim and Loh 2001; Loh 2002).

In FACT, symbolic features are converted into numeric ones, and then, for both types of features, F statistic is calculated for each variable to estimate its eligibility for the split. It favors symbolic features, especially those with many possible values.

The solution of QUEST (Loh and Shih 1997) was to reduce the bias by using different statistics (F statistic$F$ and χ^2 statisticχ^2) to estimate continuous and discrete features respectively. Final comparisons were performed for the p-values obtained from adequate distributions. Comparative bias analysis was performed for many pairs of variables of various distributions by drawing samples according to the distributions with class labels generated independently from the predictors. The scores of QUEST were quite close to the value of 0.5, expected in the case of unbiased method.

The results obtained with FACT and exhaustive search were significantly different than 0.5 in many of the tests, confirming their bias.

The feature selection strategy of QUEST was so successful that the CRUISE method of Kim and Loh (2001), published several years later, borrowed this part of the algorithm for its 1D option. Moreover, it introduced the option 2D, capable of detecting some interactions between features with negligible bias in variable selection, similarly to the univariate split analysis (1D). The 2D method, analyzing pairs of features from the point of view of contingency tables of their joint distributions, is presented in detail in Sect. 2.2.5.3.

Apart from the null case of bias analysis (assuming no discriminatory power of the predictors), Kim and Loh (2001) have also examined the problem of bias created by missing values. One of the predictors was deprived of a part of values (20, 40, 60, 80 %) to examine how it affected the split feature selection process (still assuming independence of the target from the predictors). The experiments confirmed the values of CRUISE solutions, in the sense that they selected each of the uninformative features with the same probability.

Another study of attribute selection bias was made by Shih (2004), who focused on the Pearson χ^2 statistic used in many approaches to statistical DTs like CHAID, FIRM (Hawkins 1999) and QUEST family (Loh and Shih 1997; Shih 1999; Kim and Loh 2001). The simulations performed and analyzed by Shih (2004) were composed to test two aspect of bias: the null case of class variable independent of five predictors drawn from different distributions (with some missing observations in one of the variables), and the power studies, testing the abilities to detect an informative predictor among uninformative ones. In the latter tests, different counts of predictors independent from the target were drawn as Gaussian noise. With the total of 5, 10, 15, 20 input variables, the influence of sample size on feature selection has been examined. Conclusions are compatible with those from all other similar experiments, that is, the p-values are more adequate for feature selection than χ^2 statisticχ^2 and ϕ^2 statisticϕ^2 (χ^2 divided by the number of cases) statistics.

Bias in feature selection has also been analyzed in the context of problems with multivariate responses. Some aspects of multi-label classification (where each object can belong to many classes) have been explored by Noh et al (2004), resulting in a DT induction algorithm (M2) splitting nodes with a statistic of Nettleton and Banerjee (2001) for testing equality of distributions of categorical random vectors. In M2, feature selection is also separated from split determination. Similar experiments as in other approaches (null case and power tests) are performed to confirm that the algorithm is unbiased.

An approach to learning multivariate responses by DT models has also been undertaken by Lee and Shih (2006), who generalized the variable selection method of QUEST and CRUISE for multivariate responses. They proposed *CT algorithm* based on conditional independence tests. The feature selection of CT constructs 3-way contingency tables (with dimensions corresponding to feature values, class variables, and response layers respectively) and use χ^2-test extension to estimate features with corresponding p-values. Continuous features are split into quartiles before the 3-way contingency tables are created. Lee and Shih (2006) compared their new method with an approach of Siciliano and Mola (2000), where weighted sums of Gini index reduction for each response component were used to select variables, and to the M2 algorithm of Noh et al (2004), mentioned above. Both CT and M2 are free of selection bias (in the null case), but CT is shown as providing higher estimated probability of selecting the correct covariate when the response vector depends on that covariate.

Dramiński et al. (2008, 2011) proposed a feature selection method based on a costly Monte Carlo procedure and thousands of DTs generated for different feature sets and training data objects. Feature selection based on relative importance of the features, calculated on the basis of their role in large number of trees, also shows a negligible bias.

When statistical tests are performed to select split features in DTs, there is also a simple possibility of bias reduction by means of Bonferroni corrections, respecting the numbers of possible split candidates available for particular features.

Permutation Tests

Most of the statistical approaches to feature selection for DT splits, discussed above, share the idea of preparing contingency tables and assessing independence between each feature and the target, by means of test statistics like χ^2 statisticχ^2 of G statistic$G$. They are statistically correct and successful, if the data samples, for which they are calculated, are sufficiently large. Then, both χ^2 and G are distributed with chi-squared distribution. Otherwise, the assessment accuracy may be low (Agresti 1990). Unfortunately, in DT induction, dealing with small samples is inevitable, because the data samples are getting smaller and smaller with subsequent splits.

A robust family of methods has been proposed as a result of applying permutation tests to the goal of variable assessment. The fundamental advantage of permutation tests in the context of DT learning is their eligibility for small data samples. Their p-values can be calculated directly, without passing through χ^2 or any other “transient” statistic. By analysis of all possible permutations of the series of values defining targets of the training data, one can estimate the probability of observing such input-target association as in the training data, on the assumption that the input and the target variables are independent.

Frank and Witten (1998) proposed application of approximate permutation tests presented by Good (1994). The tests are based on the multiple hypergeometric distribution. In the approach, appropriate p-values are determined and used for attribute selection and pre-pruning of decision trees.

The permutation tests framework of Strasser and Weber (1999) has been found very useful by Hothorn et al. (2004) and Zeileis et al. (2008) for another successful DT induction approach. The CTree algorithm is described in detail in section 2.2.6.

Bias in Heuristic-Based Methods

Naturally, bias in feature selection has also been a concern of the authors of heuristic based DT induction algorithms. For example, favoring discrete features with many possible values became the foundation of information gain ratio of C4.5. Also Breiman et al. (1984) noted that, when Gini gain is used as splitting criterion, "variable selection is biased in favor of those variables having more values and thus offering more splits". The analysis of Quinlan and Cameron-Jones (1995) addressed the problem of "Fluke theories" that can be selected by DT algorithms, because they seem accurate, but then, turn out to have low predictive accuracy. Analysis of multi-valued symbolic variable, means testing many theories, among which such "fluke theories" may be selected with the higher probability, the more symbols are possible in the variable. In exhaustive search, the number of tested theories grows exponentially with the number of symbols.

Kononenko (1995) analyzed a number of DT split measures in the context of multi-valued attributes, and concluded that for some of the measures, the probability of selecting a feature increases, and for some other decreases with growing number of possible feature symbols. Using p-values of statistical tests can bring almost unbiased decisions, but suffers from the problem of discriminating more and less informative attributes. When the target is certainly dependent on an attribute, the corresponding p-value gets the value of 1. Two such informative attributes get the same value, although one can still be significantly more informative than the other. When both estimates come from distributions of the same parameters (for example χ^2 with the same degrees of freedom), raw statistic values can be compared to determine which feature is better, but if the distributions are different, no tool to discriminate the two variables is available. As an alternative solution, Kononenko (1995) proposed a criterion based on the MDL principle, which is slightly biased against multi-valued attributes.

An analysis of bias of the Gini criterion used in CART has interested Dobra and Gehrke (2001), who proposed a general method to remove bias of such criteria and showed its application to Gini index. The general method of bias removal consists of two steps: first the value of split criterion is calculated and then its p-value under null hypothesis is determined. Dobra and Gehrke (2001) proposed four ways of computing the p-values for split criteria: exact computation (very expensive), bootstrap estimation (also costly), asymptotic approximations (inaccurate for small samples) and tight approximations (may be hard to find). The authors presented a tight approximation of the Gini gain.

Another p-value based measure derived from Gini index has been proposed by Strobl et al. (2005). Their selection criterion based on the Gini gain was inspired by the theory of maximally selected statistics. To calculate the criterion score, they estimated the distribution function of the maximally selected Gini gain, and calculated appropriate p-value under the null-hypothesis of no association between the target and predictor variables. To derive the exact distribution function of the maximally selected Gini gain, Strobl et al. (2005) used a combinatorial method following the ideas of Koziol (1991) to determine the distribution of the maximally selected χ^2 statistic.

Similarly to other authors studying bias of different DT criteria, Strobl et al. (2005) performed a null case experiment, where five predictors contained no information about the response variable, and power case studies, assuming that one feature was informative and contained some percentage of missing values.

Concluding Remarks

The pursuit of unbiased feature selection methods for DT induction has brought several very interesting algorithms, but they still are just some among many possible approaches and do not guarantee more accurate models, when solving particular tasks.

It might seem that provided an unbiased method for split-feature selection and equally valuable split selection algorithm, one can create a perfect algorithm, generating optimal DT models for all learning problems. The truth is different because of two reasons:

1. As discussed at the very beginning of the chapter, optimization at node level is not the same as optimization of tree models.
2. The term "unbiased" sounds almost like "perfect", but its definition is not as general as its common sense meaning, and as a result, unbiased methods are not as robust as they might seem from general statements.

To make the efforts viable, unbiased attribute selection is usually defined as preserving equal probability of selection of each feature, independent from the response variable. Theoretical proofs of unbiasedness have nice statistical foundations and are mathematically correct, but do not have so much practical value as might be expected, because good behavior for uninformative (statistically independent) features does not guarantee fair selection between informative features. Models useful in practice make decisions on the basis of informative features, so do not fit the theoretical frameworks. Experiments, confirming that unrelated features are selected equally often, are also correct, but do not explore the actual areas of interest.

As it has been pointed out above, p-values of independence tests run for informative features often are equal to 1, which makes just comparison impossible. When uninformative features are analyzed, their expected p-values are close to 0.5, and the analysis is focused on the area of the most changeable part of cumulative distribution functions, where comparisons may be accurate. When the p-values are equal to 1 and the values of statistics for different attributes are incomparable to each other, fair feature selection can not be done with such tools.

There is no single, commonly accepted definition of a measure of information about response variable, contained within an attribute. That's why so many split criteria have been proposed and no one seems definitely better than all the others. Again, the conclusion about the most reasonable strategy of the search for the best models, directs attention towards meta-learning techniques, capable of

- gathering and drawing conclusions from meta-knowledge about advantages of various algorithms in various applications,
- making predictions of potential gains resulting from running various learning machines for particular learning data,

- making algorithm-selection decisions after validation of models estimated as the most adequate.

Successful model selection is possible only with a bunch of powerful base-level learning algorithms, some meta-knowledge and robust, efficient meta-learner.

2.8 Ensembles of Decision Trees

Many researchers have been attracted by the idea of constructing complex models on the basis of collections of other models. Scientists working in the area of decision tree induction have been especially prolific in this context. The ideas of bagging, boosting and other approaches to combining decisions of sets of models are regarded by some experts as the most significant achievements of computational intelligence research of the 1990s.

Averaging decisions of multiple models can be justified in many theories. One of the most sensible way is the analysis on the ground of Bayesian learning theory. When many models (hypotheses) exist for given training data D, the optimal choice of the class $c \in \mathscr{C}$ for a data object x is defined by the *Bayes Optimal Classifier*:

$$BOC(x|D) = \arg\max_{c \in \mathscr{C}} P(c|x, D) = \arg\max_{c \in \mathscr{C}} \sum_{M \in \mathscr{M}} P(c|x, M)P(M|D). \quad (2.87)$$

Although it is usually not possible to explore the whole space $\mathscr{M}$ of possible models, approximations by some families of probable models are very sensible. Often, a single model is selected on the basis of a criterion like MAP (*maximum a posteriori*) or ML (*maximum likelihood*):

$$M_{MAP} = \arg\max_{M \in \mathscr{M}} P(M|D) = \arg\max_{M \in \mathscr{M}} P(D|M)P(M), \quad (2.88)$$

$$M_{ML} = \arg\max_{M \in \mathscr{M}} P(D|M). \quad (2.89)$$

They can be seen as extreme approximations of the $\mathscr{M}$ family by one-element sets. Informally, one can claim that such approximations make more sense than using random collections of models, because the MAP and ML models are models of confirmed quality.

More detailed analysis and more information on Bayesian learning theory can be found, for example, in the (very good) chapter on the subject by Mitchell (1997).

Approximating Bayes Optimal Classifier by a restriction of usually infinite model space $\mathscr{M}$ to a finite set of models $\mathscr{M}' = \{M_1, \ldots, M_s\} \subseteq \mathscr{M}$ and estimation of $P(M|D)$ by some weights w_M leads to a classifier estimating class probabilities for given object x as:

$$P(c|x) = \sum_{M \in \mathscr{M}'} w_M \cdot P(c|x, M). \quad (2.90)$$

Estimation of the conditional probabilities $P(c|x, M)$ of class c given DT models is quite easy (although not unanimous, so many authors have proposed their solutions). More difficult part of the task is finding proper weights w_m. Some researchers try to go further in Bayesian analysis and determine $w_m = P(M|D)$ as $P(D|M)P(M)$ (Buntine 1993), but here also the definition of priors $P(M)$ is not self-evident. In other approaches, the weights are assumed to be equal (like in bagging and other unweighted voting scenarios) or are determined by miscellaneous algorithms to reflect model competence and the strength of its influence on final ensemble decisions.

By averaging decisions of many models, one can also generalize to new variations, not observed in the training data sample (Bengio et al. 2010). This feature is unavailable to single DT learners, because the splits are determined on the basis of the evidence available in the training data.

Building complex models can bring improvement in approximation error, but a price must be paid for that. Computational costs are the most obvious, but not the only ones. In the realm of decision tree models, the most significant loss accompanying the ensemble gains in modeling accuracy is the loss of model comprehensibility. This cost is especially oppressive, because readable and understandable form is one of the most important reasons of DT induction popularity and appreciation. Interpretation of complex models is much more difficult, so different forms of visualization and explanation of the decision functions are created as some recompense.

Ensembles owe the improvement in accuracy to diversity of their component models. Only a set of committee members specializing in different subareas of the domain of learning can introduce new value, when properly combined. Model diversity may come from different sources (Zenobi and Cunningham 2001; Melville and Mooney 2003; Brown et al. 2005). Two groups of the sources seem the most important:

- different learning algorithms (methods of completely different domains or just changes in parameters of a single learning strategy),
- different training datasets (sampling, transformations and so on).

Analyzing similarity between algorithms does not seem to make much sense, as no dissimilarity guarantees diversity of resulting models. Quite often, completely different DT induction algorithms create very similar (or even exactly the same) decision trees. From formal point of view, it is not justified to distinguish between "completely different algorithms" and "small differences in parameters of the same algorithm", because even the smallest change in parameter settings results in a different algorithm. It is also quite common that a small change in a single parameter results in completely different DT model.

Decision tree induction methods are known to be unstable, which means that small changes of the input data may cause significant changes in the results of learning. Because of that, in the realm of DTs, manipulating the training sample seems much more interesting source of model diversity, and has been explored by many scientists.

In construction of ensembles, two techniques can be distinguished with respect to the fundamental organization of the member learning processes:

- *independent model generation*, also called *perturb and combine* (P&C) approach,
- *dependent subsequent models*, also referred to as *adaptive resample and combine* methods.

The former group facilitates easy parallelization of computations, because each member model is generated independently. In the latter approach, ensemble members must be generated sequentially, one by one, because each next model is constructed on the basis of the results of all the previous models.

Practical learning problems are usually defined by a restricted set of data object descriptions that can be used for learning, so that it is not possible to generate arbitrary many datasets of arbitrary size. Therefore, generation of different training datasets is not a trivial task. The most popular approaches belong to one of two groups: *resampling* and *reweighting*. Resampling methods generate new sets by selection of objects from the original dataset (possibly with repetitions). Reweighting techniques assign weights to each data object from the original dataset and pass the weights to the learning algorithm together with the dataset. Obviously, such operation makes sense only when the learning algorithm accepts and can take advantage of such weights in its learning process.

The most popular approach to independent model generation is *bagging* (Breiman 1996; Quinlan 1996) described in Sect. 2.8.2. Its original definition was a resampling method, but a modification to accept weights (called *wagging*) has also been examined (Bauer and Kohavi 1999).

The family of algorithms generating dependent models is usually referred to as *boosting* and is described in Sect. 2.8.3. The methods are so general that can be used in the manner of both resampling and reweighting.

Breiman (2001) presented a general view of DT ensembles and presented some particular algorithms for random forests generation. More on these methods can be found in a successive subsection.

Many other algorithms have also been proposed to construct ensemble models. Some authors noticed that partitioning methods like cross-validation can be successful in building diverse committees (Parmanto et al. 1995; Domingos 1996; Grąbczewski and Jankowski 2006a; Grąbczewski 2012).

Another idea to collect diverse DT models for ensemble construction is to perform more thorough search in the space of DT models and collect a number of attractive trees instead of just a single tree classifier. Beam search is a perfect tool for such purposes and has been used to induce DT forests based on SSV criterion, also heterogeneous ones, that is, using premises concerning distances from prototypes apart from standard single feature splits (Grąbczewski and Duch 2002a, 2002b).

2.8.1 Option Decision Trees

Option Decision Trees (ODT, Buntine 1993; Kohavi and Kunz 1997) are classification models comprising many DTs in a single complex structure. Actually, ODT structures are equivalent to a number of separate DTs, but thanks to keeping them together, no common branches are represented in multiple copies. During DT induction process, at each node, several alternative splits are recorded if possible, and the whole alternative branches are constructed.

Nevertheless, not the memory saving is the main focus of ODTs. Thanks to collecting alternative splits at each node of the tree, alternative paths are available and each data object can be classified with respect to many paths, by proper decision averaging. Buntine (1993) has proposed a framework for *Bayesian averaging* of collections of possible decision paths in trees. The framework is applicable also to single trees, because a set of trees resulting from all possible ways of pruning the tree can be seen as an approximation $\mathscr{M}'$ of the space of models of Eq. (2.90). With the procedure named *tree smoothing*, Buntine (1993) assigned probabilities to tree leaves that are reported to estimate class probabilities more accurately.

The most serious drawback of the approach is significant increase of the computation time in comparison to single DT induction techniques. Buntine tested his algorithms also with more thorough search techniques like n-ply lookahead search$n$-ply lookahead with beam width restriction, but naturally, it additionally increased the time of computations.

2.8.2 Bagging and Wagging

The term *bagging* comes from the expression bootstrap*bootstrap aggregating* (Breiman 1996; Quinlan 1996). Diverse learning machines are created by means of preparing *bootstrap samples* for each subsequent learning process. A bootstrap sample drawn from a given dataset D is a collection of items drawn from D at random, independently, with replacement. This shows the inadequacy of the term "dataset", as in the bootstrap sample, an object can occur several times. It has been confirmed that bootstrap samples are successful sources of model diversity.

The technique of bagging is presented by Algorithm 2.22. A predefined number of bootstrap samples of the same size as the original training data is created and a model learned from each. Eventually, the decisions of all the models are combined by ordinary majority voting.

If the learning algorithm applied to generate the ensemble member models can learn with respect to weights assigned to training data objects, instead of drawing bootstrap samples, one can just draw weights and pass them to the learning machine. Such methodology has been named *wagging* (for *weight aggregation*, Bauer and Kohavi 1999) and is presented as Algorithm 2.23.

The definition of the algorithm is very general, as it refers to arbitrary weighting distribution given as a parameter. Bauer and Kohavi (1999) added Gaussian noise to each weight with mean zero and a given standard deviation. Because negative weights do not make sense, each weight value falling below 0 is treated as 0 and the object assigned such weight has no influence on the learning process (the object is treated as nonexistent). Increasing the standard deviation of the noise reduces the

Algorithm 2.22 (Bagging)

```
Prototype: Bagging($D, s, L$)
Input: Training data $D$ containing $n$ objects, ensemble size $s$, learner (machine) $L$.
Output: Voting committee.
The algorithm:
    1. for i=1,...,s do
        a. $D_i \leftarrow$ bootstrap sample of size $n$ generated from $D$
        b. $M_i \leftarrow L(D_i)$ /* train a machine */
    2. return the committee $\{M_1, \dots, M_s\}$
```

Algorithm 2.23 (Wagging)

```
Prototype: Wagging($D, s, L, d$)
Input: Training data $D$, ensemble size $n$, learner (machine) $L$, weighting distribution $d$.
Output: Voting committee.
The algorithm:
    1. for $i = 1, \dots, s$ do
        a. $\mathbf{w} = (w_1, \dots, w_n) \leftarrow$ weights drawn randomly from $d$ for each data object in $D$
        b. $M_i \leftarrow L(D, \mathbf{w})$ /* train a machine */
    2. return the committee $\{M_1, \dots, M_s\}$
```

training dataset, so increases the bias and reduces the variance of learning, facilitating some control on the bias-variance trade-off.

When comparing the algorithms of bagging and wagging, it can be noticed that the learning process is called in different ways. This reflects the difference between resampling and reweighting that bore the wagging algorithm.

Although the original definition of bagging assumed ordinary majority voting as the final decision function of the ensemble, the decision module can be modified in several ways. For example, the combination may reflect probabilities estimation (of belonging to particular classes) returned by probabilistic classifiers as in the experiments presented in Chap. 5.

Another interesting technique to improve bagging and wagging results is *backfitting* proposed by Bauer and Kohavi (1999). Because the training data samples generated for bagging purposes, contain significantly less objects than the original data (around 63.2 %, see the explanation below), it is advantageous to feed the whole original dataset to the tree and estimate class probabilities at the leaves more accurately. Combined probabilities, estimated in this way, provide better results than simple voting and then probabilities estimated on the bootstrap samples. Similarly, in wagging, the training dataset can be passed through the trees with equal weights for all the objects to obtain better estimation of class probabilities.

Error Estimation with .632 Bootstrap

An analysis of bootstrapping in the context of bagging (Efron 1983; Efron and Tibshirani 1997) brings interesting conclusions about learning possibilities. The probability that a given data object occurs in the bootstrap sample of size n drawn from an input dataset of size n is equal to

$$1 - \left(1 - \frac{1}{n}\right)^n . \tag{2.91}$$

With $n \to \infty$ it converges to $1 - \frac{1}{e} \approx 0.632$. Already for $n = 24$ the value is less than 0.64. The larger n the closer to the limit. Hence it is justified to approximate the part of original dataset occurring in the bootstrap sample as 63.2 %.

Because bootstrap samples contain on average just 63.2 % of objects from the training set, the error estimates are pessimistic. On the other hand, it is obvious that the error estimate calculated for the data used for training can be too optimistic. Therefore, Efron (1983) proposed the *.632 estimator*:

$$\widehat{err}^{(.632)} = .368 \cdot \overline{err} + .632 \cdot \widehat{err}^{(1)}, \tag{2.92}$$

where $\overline{err}$ is the error estimate calculated from training data (biased downwards) and $\widehat{err}^{(1)}$ is the bootstrap estimate (prediction from classification error calculated for points not occurring in the bootstrap sample; biased upwards). Efron and Tibshirani (1997) suggested further improvement of the estimate and proposed a $\widehat{err}^{(.632+)}$ estimate shifting the balance between $\overline{err}$ and $\widehat{err}^{(1)}$ towards the latter, on the basis of a factor named *relative overfitting rate*.

Recent studies by Kim (2009), aimed at fair comparison of three approaches to estimating error rates of classifiers (repeated cross-validation, repeated hold-out and .632 bootstrap), have brought conclusions that different methods provide the best estimations for different tasks, depending on the learning sample size and the classification learner being tested. In particular, repeated CV turned out to be more adequate for "highly adaptive" classifiers, that is, methods like boosting, capable of gaining resubstitution error close to 0.

2.8.3 Boosting

The idea of *boosting classifiers* has been introduced by (Freund and Schapire 1995, 1996, 1997). They have proven some important properties justifying the solutions. In general, *boosting* is a method of converting "weak" learning algorithm to a "strong" algorithm with arbitrarily high accuracy. Because the main means of the method is adaptive resampling (or reweighting) and combining, Breiman has used the name *arcing* for this technique.

Boosting classifiers is a technique of repeated training of a given learning machine on data samples (or weighted data) generated with respect to probability distributions adjusted to the results of previous learning processes. After a number of models is learnt, they form a committee with properly weighted decisions.

AdaBoost Algorithm

One of the first boosting procedures proposed by Freund and Schapire (1995) was AdaBoost (for adaptive boosting). The algorithm is still the most popular one of this kind. It is presented formally as Algorithm 2.24. The method builds a collection of models on the basis of the training dataset and probabilities p_x assigned to each object x of the training set. The learning stage (item 2a of the algorithm) can be realized in different ways, depending on the preferred (or at all possible) strategy: *weighting* or *sampling*. If the learning process accepts weights assigned to each training data object, then the training dataset and the weights may be passed to it. Otherwise, a data sample is derived from the training set with respect to the probability distribution p and passed to the learning machine.

Algorithm 2.24 (AdaBoost)

```
Prototype: AdaBoost($D, s, L$)
Input: Training data $D = \{(x_1, c_1), \ldots, (x_n, c_n)\}$, ensemble size $s$, "weak" learner $L$.
Output: Weighted committee.
The algorithm:
    1. for each $(x, c) \in D$ do /* initialize the probability distribution as uniform */
           $p_x \leftarrow \frac{1}{n}$
    2. for i=1,...,s do
           a. $M_i \leftarrow Learning(L, D_i, p.)$ /* train a machine with respect to p. */
           b. $\varepsilon_i \leftarrow \sum_{\{(x,c)\in D: M_i(x)\neq c\}} p_x$
           c. if $\varepsilon_i > \frac{1}{2}$ then
                  i. $s \leftarrow i - 1$
                  ii. break the loop
           d. $\beta_i \leftarrow \frac{\varepsilon_i}{1-\varepsilon_i}$
           e. for each $(x, c) \in D$ do /* modify the probability distribution */
                  $p_x \leftarrow p_x \cdot \frac{1}{2\varepsilon_i} \cdot \beta_i^{\mathbf{1}_{\{c\}}(M_i(x))}$
    3. return the weighted committee $M = \left(\{M_1, \ldots, M_s\}, \{\log \frac{1}{\beta_1}, \ldots, \log \frac{1}{\beta_s}\}\right)$:
           $M(x) = \arg\max_{c\in\mathscr{C}} \sum_{i:M_i(x)=c} \log \frac{1}{\beta_i}$
```

The initial distribution is uniform, so D_1 is a bootstrap sample generated in the same way as in the bagging approach. Then, after each subsequent model M_i is created, its error ε_i is calculated and the probability distribution is modified (see item 2e of the algorithm) to focus the subsequent learning processes more on the misclassified data objects than on the ones classified correctly. The idea of distribution changes is to multiply by $\beta_i < 1$ (diminish) the probabilities of correctly classified objects in such a way that the sum of probabilities of incorrectly classified objects becomes equal to $\frac{1}{2}$ (the other half is the sum of probabilities of correctly classified objects). Hence the factor $\beta_i \leftarrow \frac{\varepsilon_i}{1-\varepsilon_i}$, because the sum of p_x for all correctly classified xs, before the modification is $1 - \varepsilon_i$, so after multiplication by β_i becomes ε_i which by definition is the sum for incorrectly classified objects. Therefore, the factor $\frac{1}{2\varepsilon_i}$ restores the properties of probability distribution.

The final decision of the ensemble is made on the basis of combined decisions of the members with larger weights for more accurate models and smaller for the poorer classifiers. The goal is reached by the weights specified in AdaBoost as $\log \frac{1}{\beta_i}$.

Such formulation of the AdaBoost algorithm has originally been named AdaBoost.M1 as a modification of the first formulation devoted to two-class problems. In multi-class classification, one of the most significant weak points of AdaBoost is the requirement that the errors of weak learners $\varepsilon_i < \frac{1}{2}$. Otherwise, the factors β_i would get larger than 1 and the goal of boosting would get inverted: the probabilities for erroneously classified objects would be decreased instead of increased. To remedy this, Freund and Schapire (1995) proposed an algorithm named AdaBoost.M2, which uses another shape of the models (fuzzy classification by assigning a value in the interval [0, 1] to each class instead of crisp class assignment), another scheme of handling the probability distribution and modified ensemble decision function. The algorithm is less popular, so it is not presented here in detail. It can be found in the articles by Freund and Schapire (1995, 1996, 1997).

When the error of subsequent model gets larger than $\frac{1}{2}$, the AdaBoost algorithm is stopped and the ensemble is composed of the models, found so far, without the last one of too large error (see item 2(c)ii of Algorithm 2.24). Other authors suggest restarting the probabilities (going back to the uniform distribution) in such circumstances, instead of breaking the process or preforming the normal scheme of probability adjustment. After the reset of the probability distribution, the next sample is again a bootstrap sample. If each model results in error greater than $\frac{1}{2}$, then each sample is a bootstrap sample, and the member models are the same as in bagging. The only difference with bagging, in such case, is the weighted decision function of the final model instead of the majority voting used in bagging.

When a model at some stage perfectly classifies the training data ($\varepsilon_i = 0$), the next stage of AdaBoost is not feasible, because the distribution gets degenerate. In such cases, it is also suggested by some authors to reset the probability distribution and build next model on another bootstrap sample.

AdaBoost Modifications

Many different variants of AdaBoost can be found in the literature. Three of them have been used in the experiments described in Chap. 5: *conservative boosting* (Kuncheva and Whitaker 2002), *averaged boosting* (Oza 2003) and *averaged conservative boosting* (Torres-Sospedra et al 2007).

Conservative boosting (Kuncheva and Whitaker 2002), as suggested by its name, uses a more conservative method of distribution adjustment. The conservativeness is realized by changing the step 2d of Algorithm 2.24 to

$$\beta_i \leftarrow \sqrt{\frac{\varepsilon_i}{1-\varepsilon_i}}, \tag{2.93}$$

and replacing the denominator $2\varepsilon_i$ of the renormalization factor used in step 2e to $\sum_{(x,c)\in D} p_x \cdot \beta_i^{1_{\{c\}}(M_i(x))}$. The introduction of the square root makes the factor β_i larger, so the decrease of the probabilities assigned to correctly classified objects is slower and more correctly classified objects are kept in the next training data. The change of normalization factor is a natural consequence of the modified β_i.

Kuncheva and Whitaker (2002) have also tried a version of boosting called *inversed*, because by changing the factor from $\beta_i^{1_{\{c\}}(M_i(x))}$ to $\beta_i^{1-1_{\{c\}}(M_i(x))}$, the probabilities assigned to correctly classified vectors are increased in this approach, while the incorrectly classified objects get less probable in the next sample. The inversion is a technique similar in idea, to iterative refiltering, used in the DT-SE family of DT induction methods (see section 2.3.4).

The algorithm of *averaged boosting* (Oza 2003) also differs from AdaBoost in the way it modifies the probability distribution after each step. It also slows down the changes in comparison to the standard AdaBoost, as instead of the new value calculated in the AdaBoost manner, it sets new values as the average of all AdaBoost corrected values from the first stage of the process:

$$p_x \leftarrow \frac{i \cdot p_x + p_x \cdot \frac{1}{2\varepsilon_i} \cdot \beta_i^{1_{\{c\}}(M_i(x))}}{i+1}. \tag{2.94}$$

Averaged conservative boosting (Torres-Sospedra et al 2007) is the method combining the ideas of averaged boosting and conservative boosting. The β_i factor of the algorithm is defined by Eq. (2.93), and the probability distribution is corrected according to the formula

$$p_x \leftarrow \frac{1}{Z_i} \cdot \frac{i \cdot p_x + p_x \cdot \beta_i^{1_{\{c\}}(M_i(x))}}{i+1}, \tag{2.95}$$

where Z_i is the renormalization value that guarantees the probability distribution properties of p. This definition may seem slightly inconsequent, because the average is calculated from normalized p_x and not normalized new part $p_x \cdot \beta_i^{1_{\{c\}}(M_i(x))}$. It seems more adequate to normalize the new part first and then calculate the weighted mean, which would not need further normalization as is averaged boosting.

Arc-x4 Algorithm

Breiman (1998) has also undertaken the analysis of adaptive resampling and combining (hence the term *arcing*). He has examined the original approach of Freund and Schapire (1995, 1996, 1997) to boosting in application to creating DT ensembles and introduced his own ad-hoc algorithm of similar idea. In this work, the original boosting approach was referred to as *arc-fs* and the Breiman's ad-hoc version as *arc-x4*. Arc-fs was modified to reset the uniform distribution in the case of $\varepsilon_i > \frac{1}{2}$ or $\varepsilon_i = 0$.

Algorithm 2.25 presents the Breiman's algorithm formally. The fourth power of $m_{i,x}$ was chosen as the best of three tested values (1, 2 and 4), so it does not come from a significant optimality analysis. Even so simple boosting approach turned out to be quite successful.

Algorithm 2.25 (Arc-x4)

Prototype: ***Arc-x4***(D, s, L)
Input: *Training data* $D = \{(x_1, c_1), \ldots, (x_n, c_n)\}$, *ensemble size* s, *"weak" learner* L.
Output: *Voting committee.*
The algorithm:

```
1. for each $(x, c) \in D$ do /* initialize the probability distribution as uniform */
       $p_x \leftarrow \frac{1}{n}$
2. for $i = 1, \ldots, s$ do
       a. $M_i \leftarrow Learning(L, D_i, p.)$ /* train a machine with respect to p. */
       b. for each $(x, c) \in D$ do /* modify the probability distribution */
              $p_x \leftarrow \frac{1 + m_{i,x}^4}{\sum_{(x',c') \in D} 1 + m_{i,x'}^4}$,
          where $m_{i,x}$ is the number of models in $\{M_1, \ldots, M_i\}$ that misclassify $x$
3. return the voting committee $M = (M_1, \ldots, M_s)$
```

Breiman used resampling to generate each training dataset D_i on the bases of the probability distribution p. Apart from the training set, he used another set generated in the same way, from the same probabilities p, as a validation set for DT pruning. It was significantly more efficient than pruning on the basis of cross-validation.

TreeNet

Another boosting approach worth mentioning is a commercial product named *TreeNet*. It arose from the *Multiple Additive Regression Tree* (MART) system dated back to 1999. The fundamental idea of the approaches is the *stochastic gradient boosting* technology of Friedman (1999a,b). Similarly to the boosting approaches described above, also here, the ensembles are additive models combining a set of base models:

$$M(x) = \sum_{i=1}^{s} \beta_i M_i(x). \tag{2.96}$$

Each subsequent model M_i is selected to minimize the value of a loss function, determined on the basis of a training dataset. This technique is a very similar to *cascade correlation* used for training neural networks, where neurons are added in sequence to improve the performance of the network on the training set.

The main difference between the methods of MART (or TreeNet) and AdaBoost is that the gradient boosting methods are devoted to regression, not classification problems. Naturally, classification tasks can be solved by means of regression tools with binomial log-likelihood loss function.

The power of the algorithms is in large numbers of DT models combined. Each single tree learns very little from the data. It can be claimed that high quality of single DTs is not desired in these approaches. Because of that, the trees are never trained on the whole training dataset—usually a random half of the data is used for learning a single tree.

Single trees are not very adequate models for regression goals, because they represent coarse step functions (piecewise constant). Thanks to combining large numbers of trees, quite "smooth" curves can be obtained. Because of large numbers of combined models, the final models do not share the advantages of single tree comprehensibility. Instead, special reports are designed to extract the meaning of the model in the form of feature importance rankings or graphs illustrating the relationship between inputs and outputs.

Alternating Decision Trees

Induction of a generalized decision trees called *alternating DTs* (ADTrees) has been proposed by Freund and Mason (1999).

As a generalization of DTs, ADTrees can represent standard DTs but also significantly more complex structures. They have a form very similar to the one of Option Decision Trees. Similarly to ODT, they can encode many trees in a single structure. Thus, they can also easily represent voting stumps.

ADTrees consist of two types of nodes (*prediction nodes* and *decision nodes*) that occur interchangeably on tree paths. Each prediction node is assigned a real value used in decision making on the basis of the structure: all the real values on the multi-path traversed by an object to be classified are summed and the sign of the sum decides about classification into one of two classes. An object can follow a multi-path, not a single standard path, because in a general alternating tree, at each prediction node, all children are tested and point different further paths.

Decision (splitting) nodes of ADTrees are described with a special form of rules, where *precondition* corresponds to the path from the root to the decision node and *condition* represents the test of the node.

To learn ADTree models, the authors decided to use a modification of Adaboost algorithm proposed by Schapire and Singer (1999), because it is suitable for dealing with real valued predictions of ADTrees.

Freund and Mason (1999) declared that their approach showed results competitive with boosted C5.0 trees, but usually with smaller and easier to interpret trees. As another important advantage, they pointed out that alternating trees give a natural measure of classification confidence.

2.8.4 Random Forests

Breiman (2001) proposed a general definition of decision tree forests as a collection of tree classifiers built with respect to random vectors. In the framework, given a random vector Θ_i for each $i = 1, \ldots, s$, a tree is grown for the training data and Θ_i. Denoting i'th DT classification model as $M_{\Theta_i} : O \rightarrow \mathscr{C}$, a *random forest* is defined as the majority voting classifier based on the collection of DT models $\{M_{\Theta_i} : i = 1, \ldots, s\}$. Additionally, Breiman (2001) assumed that the random vectors Θ_i were independent and identically distributed.

Such definition of random forests encircles many different schemes of DT ensembles, for example bagging, where the random vectors Θ_i may directly correspond to the n (n is the number of elements in the training dataset) object indices pointing the elements of subsequent bootstrap sample.

Boosted classifiers might also fit the definition of random forests, but the assumption of independent and identically distributed Θ_i is not satisfied. To be precise, we may state that boosting algorithms do not conform to the idea of random forest construction, although each particular boosted model might be obtained with the scheme.

The definition of random forests is constructed in a way suggesting the algorithm for growing forests. Such algorithm, for subsequent values of i, draws the random vector Θ_i and then uses it in the learning process to obtain model M_{Θ_i}.

Randomization of the trees composing forests may come from different sources, for example, training data sampling or different configurations of the DT inducer. Breiman (2001) analyzed two methods of random feature selection for each DT split within the CART algorithm:

- random input selection consisting in drawing a small group of input variables (pre-specified count F) from the set of all features describing data objects and limiting the analysis of possible splits to the F selected variables only,
- random linear combinations of randomly selected inputs, where F linear combinations are generated and analyzed to find the best split; each combination is determined by random selection of features to combine (of pre-specified size L) and drawing L coefficients from uniform distribution on $[-1, 1]$.

The randomization idea of random forests was inspired by the article of Dietterich (2000), where C4.5 algorithm was modified to randomize split selection at each DT node. The idea was to introduce diversity by means of split randomization without counting on inducer instability. For each node, the split was randomly selected from the 20 best splits determined in the normal way. The candidate splits were not pre-selected in any way, so in special cases, all 20 best splits could involve the same (continuous) attribute.

Random feature selection used by Breiman (2001) has additional advantage of reducing the cost of computations needed for DT induction. The approach of Dietterich (2000) required the same calculations as normal C4.5 run plus insignificant time for random selection of the split at each node. In the case of random input selection, calculations may be much cheaper, because not all features are analyzed, but only the selected ones. Therefore random forests can easily handle data with large number of attributes—in practice, the complexity does not depend on the number of features composing the object space. The speed up and improved scalability certainly belong to the most attractive properties of the random forests approach.

2.9 Other Interesting Approaches Related to DT Induction

Many more (than described above) algorithms for DT induction have been proposed by miscellaneous authors. It is not possible to present all the techniques in a single article or even book. Many comparative analyses have been performed and are certainly worth a focus, when searching for interesting solutions. Some reviews and comparisons have been published by Safavian and Landgrebe (1991), Murthy (1998), Provost and Kolluri (1999), Anyanwu and Shiva (2009), Rokach and Maimon (2010), and Kotsiantis (2011).

Miscellaneous Split Quality Measures

The most commonly used and some other interesting split quality measures have been presented above. Exhaustive discussion of all other methods published by CI researchers is not possible in this book, but it would not be right to completely ignore all of them. Therefore, some measures are shortly listed below. Some comparisons of selected criteria are available in the literature. For example, Kononenko (1995) compared eleven measures and introduced the MDL measure presented in Sect. 2.4.2.4. Beside the well known measures of Gini index and information gain ratio, he examined the methods of *J-measure*, *Mántaras distance*, *average absolute weight of evidence*, *relief*, *relevance*, measures based on χ^2 statisticχ^2 and G statistic$G$ statistics and the proposed MDL measure.

J-measure
(Goodman and Smyth 1988b) was defined for discrete random variables X and Y, in the language of information theory:

$$J(X|Y = y) = P(y) \sum_x P(x|y) \cdot \log \left(\frac{P(x|y)}{P(y)} \right). \tag{2.97}$$

It has been used in the ITRULE system (Goodman and Smyth 1988c; Smyth and Goodman 1992) for rule induction from data and then applied also to DT induction

(Goodman and Smyth 1988a). J-measure has also been applied to pre-pruning of DTs, resulting in a method named *J-pruning* Bramer (2002).

Mántaras distance
Similarly to information gain and J-measure, Mántaras distance is also defined in the language of information theory. With the definitions of entropy, joint entropy and conditional entropy as in formulae (1.8) and (1.9) in the introduction, the information gain, resulting from the split according to feature A can be written as

$$IG(A, C) = H_C - H_{C|A} = H_C + H_A - H_{A,C}. \tag{2.98}$$

Mántaras distance between partitions determined by feature A and class C can be expressed as

$$MD(A, C) = 1 - \frac{IG(A, C)}{H_{A,C}}. \tag{2.99}$$

Absolute weight of evidence
The measure of *absolute weight of evidence* has been introduced by Michie (1990) for two-class problems, but it can be easily extended to multi-class problems by averaging:

$$WE(A, C) = \sum_{c\in\mathscr{C}} p_c \sum_{a\in\mathscr{A}} p_{a\cdot} \left|\log \frac{p_{c|a}(1 - p_c)}{(1 - p_{c|a})p_c}\right|, \tag{2.100}$$

where $p_{c|a}$ is the proportion of objects of class c among those with value a of A.

Relief
The *relief* algorithm was designed to solve feature selection problems (Kira and Rendell 1992a,b). It adjusts weights assigned to the features (initially zeros for all features) on the basis of an iterative procedure, which for each data vector finds its closest positive and negative instances (called nearest-hit and nearest-miss adequately to the class of the data vector) and increases the weights of features "responsible" for the distance to nearest-miss and decreases the weights of features participating in the distance to the nearest-hit (the larger the distance the bigger the change). An extension of the algorithm has been proposed, that respects k nearest hits and misses in the analysis (*Relief-A*). Kononenko (1994) analyzed the algorithm and proved that if the k is not restricted (all data vectors are taken into account), then the weight for a discrete feature A (with values in $\mathscr{A}$) is highly correlated with Gini index:

$$\mathit{Relief}(A, C) = \frac{\sum_{a\in\mathscr{A}} p_{a\cdot}^2}{\sum_{c\in\mathscr{C}} p_{\cdot c}^2(1 - \sum_{c\in\mathscr{C}} p_{\cdot c}^2)} \times \mathit{Gini'}(A, C), \tag{2.101}$$

where

$$\mathit{Gini'}(A, C) = \sum_{a\in\mathscr{A}} \left(\frac{p_{a\cdot}^2}{\sum_{a\in\mathscr{A}} p_{a\cdot}^2} \sum_{c\in\mathscr{C}} p_{c|a}^2 \right) - \sum_{c\in\mathscr{C}} p_{\cdot c}^2, \tag{2.102}$$

which differs from Gini index in that it uses coefficients $\frac{p_{a\cdot}^2}{\sum_a p_{a\cdot}^2}$ instead of Gini's $p_{a\cdot} = \frac{p_{a\cdot}}{\sum_a p_{a\cdot}}$.

Relevance index
Baim (1988) introduced the following relevance measure of a partition A with respect to classes C:

$$Relevance(A, C) = 1 - \frac{1}{|\mathscr{C}| - 1} \sum_{a \in \mathscr{A}} \sum_{\substack{c \in \mathscr{C} \\ c \neq c_m(a)}} \frac{n_{ac}}{n_{\cdot c}}, \tag{2.103}$$

where $c_m(a) = \arg\max_{c \in \mathscr{C}} \frac{n_{ac}}{n_{\cdot c}}$. The purpose of the measure was feature selection, so it perfectly fits the needs of split quality evaluation.

ORT
ORT criterion, introduced by Fayyad and Irani (1992a) measures the quality of binary splits on the basis of orthogonality of class probability vectors calculated for the two data subsets resulting from the split. For a data sample D and a test τ inducing a binary partition on D into D_τ and $D_{\neg\tau}$, having class probability vectors $\mathbf{V}_\tau$ and $\mathbf{V}_{\neg\tau}$, respectively, the orthogonality measure is defined as

$$ORT(\tau, D) = 1 - \cos\phi(\mathbf{V}_\tau, \mathbf{V}_{\neg\tau}) = 1 - \frac{\mathbf{V}_\tau \circ \mathbf{V}_{\neg\tau}}{||\mathbf{V}_\tau|| \cdot ||\mathbf{V}_{\neg\tau}||}, \tag{2.104}$$

where $\circ$ is the inner (dot) product of two vectors. A comparative analysis of the ORT criterion and impurity measures defined as concave-maximum criteria has been presented by Crémilleux et al. (1998).

Kolmogorov-Smirnov distance
Kolmogorov-Smirnov distance between two distributions f_1 and f_2 is defined as the maximum distance between their cumulative distribution functions F_1 and F_2:

$$D(f_1, f_2) = \max_x |F_1(x) - F_2(x)|. \tag{2.105}$$

Although in classification problems, the conditional distributions of the classes with respect to the features describing the data are usually unknown, they can be estimated on the basis of the data and used as decision rules for recursive partitioning (Friedman 1977; Utgoff and Clouse 1996). Kolmogorov-Smirnov criterion for interval valued variables has been studied by Mballo and Diday (2006). They have also compared the criterion to Gini index and entropy-based decisions.

DKM criterion
Dietterich et al. (1996) and Kearns and Mansour (1999)DKM criterion explored the properties of concave functions on [0, 1], symmetric about 0.5 with maximum value 1 for argument 0.5 and minimum value 0 at the borders of the interval. The functions can play the role of impurity measures in impurity-based split criteria

(see Eq. (2.4)). The criterion based on one of the functions, $f(q) = 2\sqrt{q(1-q)}$, has been later named a DKM criterion.

Hellinger distance

Cieslak and Chawla (2008) adapted the Hellinger distance for the purpose of DT induction (HDDT). The Hellinger distance measures the divergence between two continuous distributions P and Q with respect to a parameter λ as

$$d_H(P, Q, \lambda) = \sqrt{\int_\Omega \left(\sqrt{P} - \sqrt{Q}\right)^2 d\lambda}. \tag{2.106}$$

MAP DT model

A Bayesian approach to DT induction has been proposed by Voisine et al. (2009). Their criterion is used in an optimization process performing search for maximum a posteriori (MAP) DT model. They claim that the algorithm offers similar predictive accuracy as the state-of-the-art DT induction methods, with significantly simpler trees.

CCP

After an analysis of some weaknesses of C4.5 and CART, Liu et al. (2010) proposed new measure named *Class Confidence Proportion* (CCP) and CCPDT algorithm for learning DTs on the basis of the new criterion. To provide statistically significant decision rules, they check the significance of tree branches with Fisher's exact test to decide whether to prune them.

AID Family

One of the first DT induction algorithms with statistical foundations is *Automatic Interaction Detection* (AID) proposed by Morgan and Sonquist (1963a,b). In AID trees, at each binary split the between-group-sum-of-squares (the F statistic) is maximized for each predictor, with respect to the groups determined by the dependent variable. In the case of input variables with ordered categories (called monotonic), the splits respect the ordering, while for purely nominal predictors (called free), all possible binary splits are analyzed. This results in a bias in favor of nominal features with large numbers of possible values, as they provide more possible splits to be analyzed.

An attempt to nullify the bias brought the CHAID (*Chi-Squared Automatic Interaction Detection*) algorithm (Kass 1980). Reduction of the bias of AID was achieved by significance testing and using χ^2 statistic.

The original definition of CHAID was applicable to nominal dependent variables. An extension to ordinal target variables has been proposed by Magidson (1993). Moreover, the technique of merging predictor categories with the same prediction of the dependent variable facilitated building smaller and more comprehensible DTs.

ID3 Descendants

Especially in the earliest decades of DT research, various modifications of the ID3 algorithm have been published. An example extension is the system NewID (Niblett

1989; Boswell 1990), using information gain criterion for feature and split selection (as in ID3) and facilitating analysis of continuous attributes (with similar technique as in C4.5). According to the assumptions of Niblett (1989), the NewID system was to operate with a number of split criteria, with the possibility to select the most adequate one for the data at hand. In practice, to the best of my knowledge, it has never been accomplished.

Other interesting examples of ID3 extensions are ID4 (Schlimmer and Fisher 1986) and ID5R (Utgoff 1989, 1994) systems, facilitating incremental induction.

Miscellaneous Software

Many ideas estimated by their authors as valuable have been followed by complete software solutions. Again, it is not possible to address all of them here, but some arbitrary selection of systems, capable of DT induction, is shortly commented below.

IND package
IND is a popular system written in C and C shell languages (Buntine and Caruana 1992) and distributed as opensource. It reimplements such algorithms as ID3, C4, CART and various MML (*Minimum Message Length*) and Bayesian approaches to DT induction.

1R
An idea of very simple classifiers built from a single decision rule (1R) was presented and tested by Holte (1993). The simple rules can be seen as decision trees with just a single split (so called *decision stumps*). It turns out that for many datasets tested there, so simple method offers similar accuracy as much more complex models. Naturally, there are also many datasets for which so simple models as decision stumps are significantly less attractive than more advanced solutions.

TDDT
Top-Down Decision Trees (TDDT) is the name of DT induction algorithm implemented by Kohavi et al. (1996) as a part of MLC++ (*Machine Learning in C++*) library, that became a part of the SGI's MineSet system as *SGI MLC++*. TDDT is very similar to C4.5. The most important difference is a change in the split quality measure, which is the information gain divided by the logarithm of the number of subnodes generated by the split. The goal of the change was to remove the bias in favor of splits into many small subnodes (see Sect. 2.7 for more information on methods dealing with the bias).

SLIQ
The SLIQ algorithm (*Supervised Learning in Quest*, Quest was the name of a Data Mining project developed at IBM Almaden Research Center) was created with special emphasis on its scalability (Mehta et al. 1996). The improvement in learning time was achieved with the techniques of pre-sorting and breadth-first growth of the trees. They proposed to sort the training data items once, at the beginning of the process, according to each ordered feature, to avoid the necessity of sorting at each tree node. The algorithm deals efficiently with large disk-resident

training data, although does not get rid of the limits completely, since some data structures that must be kept in memory grow with the size of the training data. SLIQ uses Gini index for split selection and three possible methods of pruning based on the MDL principle.

SPRINT

Speed and scalability were also the most important objectives of the approach of Shafer et al. (1996), which resulted in the *SPRINT* (*Scalable PaRallelizable INduction of decision Trees*) algorithm. It was designed to remove all the memory restrictions (as compared to SLIQ, created in the same research group) and to facilitate parallelization. Shafer et al. (1996) presented both serial and parallel versions of SPRINT and the results of performance evaluation. SPRINT can be run with any impurity based split criteria.

RPart

The *RPart* package (Therneau and Atkinson 1997) is in fact a reimplementation of the ideas of CART described by Breiman et al. (1984).

RainForest

Gehrke et al. (1998, 2000) created a general framework, that can be used with many specific DT induction methods (C4.5, CART, CHAID and others) to make them fast and scalable. The authors claim that the framework offers performance improvements of over a factor of five over the SPRINT algorithm.

PUBLIC

Rastogi and Shim (2000) proposed to integrate the two stages of standard DT induction approaches, construction and pruning, in a single procedure, to fasten the induction process. The pruning procedure founded on the MDL principle makes it possible to estimate if a tree node would certainly be pruned and save some time thanks to resigning from further analysis of such a node.

BOAT

Bootstrapped Optimistic Algorithm for Tree Construction (BOATGehrke et al. 1999) is another general framework, applicable to a wide range of split selection methods. The main idea of the method is to construct an initial tree using a small subset of the training data and then refine it appropriately. It is guaranteed that the refined tree is exactly the same as the tree that would result from traditional ways of induction. The methodology has an interesting possibility of incremental update to both insertions and deletions over the training dataset.

WhiBo

WhiBo (Delibasic et al. 2011) is an open source platform for white box (component based) machine learning algorithm design. It has been created as an extension of the RapidMiner system (Mierswa et al. 2006). The goal of the framework is to facilitate algorithm design from reusable components that can be extracted from different successful algorithms. WhiBo offers a set of reusable components, including many modules for DT induction.

Miscellaneous Goals and Techniques

Even so well-defined models like decision trees can be analyzed and optimized in many various ways. Some researchers are especially interested in some particular features of the models, so they focus on these aspects and are not interested in penetration of other contexts. This usually results in general techniques, focused on particular subgoals and applicable to many kinds of components, responsible for other subgoals of DT induction.

Probability estimation
It is sometimes very important to obtain good estimates of probabilities of belonging to candidate classes, when trying to classify new data items with a DT classifier. Focus on probabilities made some authors refer to their models as *Probability Estimation Trees* (PETs). In most applications, class probabilities are derived from DTs by calculating appropriate data proportions with possible modifications like Laplace correction or m-probability-estimation (Cestnik 1990; Cestnik and Bratko 1991). Section 2.6 presents some detailed information about the most popular methods. A comparison of several algorithms with respect to probabilistic classification has been conducted by Fierens et al (2005). An approach to dealing with uncertainty during both DT induction phase and in classification with ready trees was published by Jenhani et al. (2008). They introduced a *Non-Specificity based Possibilistic Decision Tree* (NS-PDT) algorithm and its extended version, capable of building option trees. Zhang and Su (2006) analyzed probabilities from the perspective of classifiers yielding large AUC. They proposed a new AUC-based algorithm for learning conditional independence trees (CITrees).

Multivariate responses
multivariate responses In standard classification problems, there is a single discrete response variable. Some applications concern recognition of several aspects in parallel, for example in health-related problems, where several separate but correlated health problems are to be recognized. Because of correlations, it is often more appropriate to learn them together than to treat the problems as completely separate and independent. Zhang (1998) used a generalized entropy criterion and two other measures, to deal with multiple binary responses. Another set of split criteria to grow decision trees with multivariate responses has been proposed by Siciliano and Mola (2000). The new split rules were derived as extensions of criteria used in two-stage binary segmentation. An unbiased method for induction of multi-label classification trees has been presented by Noh et al (2004). As in most statistical approaches interested in unbiased feature selection, they separated the process of variable selection from the search for the best split point. They applied a test statistic to examine the equality of distributions of multi-label target variable. Another interesting generalization is the approach of Lee and Shih (2006), who realized similar ideas as in CRUISE (Kim and Loh 2001), but with the analysis of 3-dimensional contingency tables.

Constrained DT induction
Garofalakis et al. (2000, 2003) grappled with constructing decision trees satisfying additional constraints on tree size and accuracy. With such approaches, one can, for example, define the goal as obtaining properly simple trees of appropriately high accuracy. More general constraints have been analyzed by Nijssen and Fromont (2010), who proposed an adaptation of extensively studied methods from the area of local pattern mining to the field of DT induction.

Three-stage DT induction
Cappelli et al (2002) came up with an interesting idea to introduce third stage into standard DT induction processes. After tree construction and validation, they proposed procedures to guide the search for the parts of tree structure that are statistically significant, and eventually make the final tree statistically reliable.

Hybrid methods
Seewald et al. (2000) presented a hybrid learning algorithm, defined by simple modifications of C4.5 algorithm, aimed at local adjustment of inductive bias by proper selection of leaf-models. In the reduced error post-pruning of C4.5 trees, they were replacing the original leaf nodes by more sophisticated learning models like Naive Bayes or instance-based learners. Their experiments confirmed that such strategies improved performance of created models.
Generalization abilities of a classifier are much larger, when the decision borders are not restricted to perpendicular to feature space axes, as in classical DT induction approaches. Linear combinations of features are more robust, but still can not simply describe nonlinearities. Therefore, Duch and Grąbczewski (2002) proposed heterogeneous trees, that is, trees with split tests referring to distances from some points in the feature space. Naturally, the price for more flexibility is increased computational complexity of the method.
Mining data streams
Wozniak (2011) addressed the problem of learning DTs, when new data streams frequently arrive and dependencies between feature values and classes are continually changing. These are significantly different learning circumstances than in the standard approach with static training data.
Recently, Rutkowski et al. (2012) have proposed to use McDiarmid?s bound instead of the commonly used Hoeffding?s bound, to build DTs for data streams, that are almost identical with the result of standard DT induction procedures. They have examined the bounds in application to the information gain criterion and Gini index.

Uplift modeling
Rzepakowski and Jaroszewicz (2012) have explored the possibilities of using DT induction algorithms for uplift modeling, that is, prediction of changes in class probabilities caused by an action. The main goal of their approach has been to recognize data objects for which an action causes the most significant change, for example, to find out the customers that are most likely to respond to a marketing action. From another point of view, the algorithms can help in deciding which

action (of several options) to take in order to maximize profit. For this purpose, Rzepakowski and Jaroszewicz (2012) have proposed some new splitting criteria and pruning methods.

Large data analysis and induction speedup
Learning from high-dimensional data or large collections of data objects is also a very important aspect of DT induction. Some algorithms mentioned above (like SLIQ, SPRINT and RainForest) address these problems either directly or indirectly. One of the means is parallelization. Srivastava et al. (1999) described two standard methodologies: *Synchronous Tree Construction* and *Partitioned Tree Construction*, to propose a hybrid method joining the advantages of both.
Amado et al (2001) also derived their new parallel implementation of C4.5 from an analysis of different approaches to parallelization.
The CLOUDS algorithm (*Classification for Large or OUt-of-core DataSets*, Alsabti et al. 1998) samples split points of ordered features to reduce complexity of the search, which is then run in a reduced area, not exhaustively.
Beside the methods dealing with large data collections, also other techniques have been proposed to prevent too complex calculations in DT induction processes. For example, Coppersmith et al. (1999) suggested some techniques to avoid analysis of all possible partitions of nominal attributes. Because, in general, the problem of finding optimal partition is very complex, they also proposed a new heuristic search algorithm based on ordering the attribute values according to corresponding class probabilities.
Li et al. (2008) developed a clustering-based classification technique *Automatic Decision Cluster Classifier* (ADCC) for high-dimensional data. In the method, a tree clustering model is generated and Anderson-Darling test is used to automatically determine the adequate size of the resulting model. The test procedures eliminate the necessity of visual cluster validation required in a former approach of Huang et al. (2000).

2.10 Meta-Learning Germs

So far, meta-knowledge analysis has usually been performed by human experts, as it is not easy to define standard automated ways that would regularly bring valuable conclusions.

Each comparison of alternative techniques, analysis of algorithm eligibility for particular kinds of problems, and drawing conclusions about influence of learning parameters on the results can be called meta-learning.

Numerous articles discussing the subjects of various split quality measuresplit quality measures (Mingers 1989b; Buntine and Niblett 1992), different techniques of pruning methods (Quinlan 1987; Mingers 1989a; Mehta et al. 1995; Malerba et al. 1996; Esposito et al. 1997; Breslow and Aha 1997; Kononenko 1998) or advantages of using less and more complex search processes for DT induction (Quinlan and Cameron-Jones 1995; Segal 1996; Janssen and Fürnkranz 2009) are very precious for the area of meta-learning, although they should be rather treated as a prelude to actual science of meta-learning. The early approaches point the need for meta-learning solutions, as they show that various methods outperform others in various applications, but there are no simple rules that could easily point the most successful methods for a dataset at hand.

Bias analysis approaches (Kononenko 1995; Kim and Loh 2001; Shih 2004; Lee and Shih 2006; Hothorn et al. 2006b) bring some interesting conclusions and new attractive algorithms, but they do not solve the problem of algorithm selection. Although the methods are nicely supported by theoretical reasoning, they have two major disadvantages:

1. They deal with "abstract" definitions, not too compatible with most practical applications, because the bias is defined and verified for features with no information about the target, that is, features that are completely useless for classification. It does not guarantee proper feature selection when information is present in the data. Moreover, "proper feature selection" is not even defined in such circumstances.
2. As discussed in the beginning of Chap. 2, improving feature selection at each single tree node does not guarantee the best quality of the whole resulting tree. Optimization of the whole tree is impossible even for very small structures and not too large sets of features, because the analysis of feature interaction is very complex.

Techniques used by Option Decision Trees (see Sect. 2.8.1 and Buntine 1993; Kohavi and Kunz 1997) can be certainly regarded as meta-learning germs, because the analysis on meta-level, they perform, leads to proper DT ensemble with adequately combined decisions.

Another example of incorporating meta-learning in the process of DT induction is the method of *Omnivariate Decision Trees* (Yildiz and Alpaydin 2001, 2005b; Yildiz 2011), where the kind of decision borders (axis-perpendicular, linear, nonlinear) is tuned automatically on the basis of meta-analysis.

In the area of rule induction, very close to DT induction, interesting meta-learning approaches can also be found. An example is the effort of Janssen and Fürnkranz (2007, 2008, 2010), who investigated the possibility of learning rule induction heuristic from experience. The tools used for this purpose are very similar to those of the METAL project described in more detail in Sect. 6.1.4.

References

Agresti A (1990) Categorical data analysis. John Wiley & Sons, New York

Almuallim H (1996) An efficient algorithm for optimal pruning of decision trees. Artif Intell 83(2):347–362. doi:10.1016/0004-3702(95)00060-7

Alsabti K, Ranka S, Singh V (1998) Clouds: a decision tree classifier for large datasets. Tech. rep, Electrical Engineering and Computer Science, Syracuse

Amado N, Gama J, Silva FMA (2001) Parallel implementation of decision tree learning algorithms. In: Proceedings of the 10th Portuguese conference on artificial intelligence on progress in artificial intelligence, knowledge extraction, multi-agent systems, logic programming and constraint solving. Springer, London, UK, EPIA '01, pp 6–13. http://dl.acm.org/citation.cfm?id=645378.651223

Amasyali MF, Ersoy OK (2008) Cline: a new decision-tree family. IEEE Trans. Neural Netw. 19(2):356–363

Anyanwu M, Shiva S (2009) Comparative analysis of serial decision tree classification algorithms. Int J Comput Sci Secur 3(3):230–240

Baim P (1988) A method for attribute selection in inductive learning systems. IEEE Trans Pattern Anal Mach Intell 10(6):888–896. doi:10.1109/34.9110

Bauer E, Kohavi R (1999) An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Mach Learn 36:105–139. doi:10.1023/A:1007515423169

Bengio Y, Delalleau O, Simard C (2010) Decision trees do not generalize to new variations. Comput Intell 26(4):449–467. doi:10.1111/j.1467-8640.2010.00366.x

Berger J (1985) Statistical decision theory and bayesian analysis. Springer, New York

Bobrowski L (1991) Design of piecewise linear classifiers from formal neurons by a basis exchange technique. Pattern Recogn 24(9):863–870. http://www.sciencedirect.com/science/article/pii/003132039190005P

Bobrowski L (1999) Data mining procedures related to the dipolar criterion function. In: Applied stochastic models and data analysis-quantitative methods in business and industry society, Lisboa, pp 43–50

Bobrowski L (2005) Eksploracja danych oparta na wypuklych i odcinkowo-liniowych funkcjach kryterialnych. Wydawnictwo Politechniki Bialostockiej, Białystok

Bobrowski L, Krtowski M (2000) Induction of multivariate decision trees by using dipolar criteria. In: Zighed DA, Komorowski J, Zytkow JM (eds) Principles of data mining and knowledge discovery: 5th European Conference: PKDD'2000. Lecture Notes in Computer Science. Springer Verlag, Berlin, pp 331–336

Bohanec M, Bratko I (1994) Trading accuracy for simplicity in decision trees. Mach Learn 15:223–250. doi:10.1007/BF00993345

Boswell R (1990) Manual for NEWID version 2.0. Tech. rep.

Bramer M (2002) Pre-pruning classification trees to reduce overfitting in noisy domains. In: Proceedings of the third international conference on intelligent data engineering and automated learning. Springer, London, UK, IDEAL '02, pp 7–12. http://dl.acm.org/citation.cfm?id=646288.686755

Brandt S (1998) Analiza Danych. Wydawnictwo Naukowe PWN, Warszawa, tum. L. Szymanowski

Breiman L (2001) Random forests. Mach Learn 45:5–32. doi:10.1023/A:1010933404324

Breiman L, Friedman JH, Olshen A, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (1998) Arcing classifiers. Ann Stat 26(3):801–849

Breslow LA, Aha DW (1997) Simplifying decision trees: a survey. Knowl Eng Rev 12(1):1–40. doi:10.1017/S0269888997000015

Brodley CE, Utgoff PE (1992a) Multivariate decision trees. Tech. Rep. 92–82, Department of Computer Science, University of Massachusetts

Brodley CE, Utgoff PE (1992b) Multivariate versus univariate decision trees. Tech. Rep. 92–8, Department of Computer Science, University of Massachusetts

Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. J Inform Fusion 6:5–20

Buntine W (1993) Learning classification trees. Stat Comput 2:63–73. doi:10.1007/BF01889584

Buntine W, Caruana R (1992) Introduction to IND version 2.1 and recursive partitioning. Tech. rep., Moffet Field, CA. http://ti.arc.nasa.gov/opensource/projects/ind/

Buntine W, Niblett T (1992) A further comparison of splitting rules for decision-tree induction. Mach Learn 8:75–85. doi:10.1007/BF00994006

Cappelli C, Mola F, Siciliano R (2002) A statistical approach to growing a reliable honest tree. Comput Stat Data Anal 38(3):285–299

Cestnik B (1990) Estimating probabilities: a crucial task in machine learning. In: Proceedings of the ninth european conference on artificial intelligence, pp 147–149

Cestnik B, Bratko I (1991) On estimating probabilities in tree pruning. In: Kodratoff Y (ed) Machine Learning - EWSL-91. Lecture Notes in Computer Science, vol 482. Springer, Berlin, pp 138–150. doi:10.1007/BFb0017010

Cherkassky V, Mulier F (1998) Learning from data. Adaptive and learning systems for signal processing, communications and control. John Wiley & Sons, Inc., New York

Cieslak D, Chawla N (2008) Learning decision trees for unbalanced data. In: Daelemans W, Goethals B, Morik K (eds) Machine learning and knowledge discovery in databases. Lecture Notes in Computer Science, vol 5211. Springer, Berlin, pp 241–256. doi:10.1007/978-3-540-87479-9_34

Coppersmith D, Hong SJ, Hosking JR (1999) Partitioning nominal attributes in decision trees. Data Mining Knowl Discov 3:197–217. doi:10.1023/A:1009869804967

Crémilleux B, Robert C, Gaio M (1998) Uncertain domains and decision trees: Ort versus c.m. criteria. In: International conference on information processing and management of uncertainty in knowledge-based systems, pp 540–546

de Mántaras RL (1991) A distance-based attribute selection measure for decision tree induction. Mach Learn 6:81–92. doi:10.1023/A:1022694001379

de Sá JPM (2001) Pattern recognition. Concepts, methods and applications. Springer, Berlin

Delibasic B, Jovanovic M, Vukicevic M, Suknovic M, Obradovic Z (2011) Component-based decision trees for classification. Intelligent Data Analysis, pp 671–693

Dietterich TG (2000) An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Mach Learn 40:139–157. doi:10.1023/A:1007607513941

Dietterich T, Kearns M, Mansour Y (1996) Applying the weak learning framework to understand and improve C4.5. In: Proceedings of the thirteenth international conference on machine learning. Morgan Kaufmann, pp 96–104

Dobra A, Gehrke J (2001) Bias correction in classification tree construction. In: Brodley CE, Danyluk AP (eds) Proceedings of the eighteenth international conference on machine learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001, Morgan Kaufmann, pp 90–97

Doetsch P, Buck C, Golik P, Hoppe N, Kramp M, Laudenberg J, Oberdörfer C, Steingrube P, Forster J, Mauser A (2009) Logistic model trees with auc split criterion for the kdd cup 2009 small challenge. J Mach Learn Res Proc Track 7:77–88

Domingos P (1996) Using partitioning to speed up specific-to-general rule induction. In: Proceedings of the AAAI-96 workshop on integrating multiple learned models. AAAI Press, pp 29–34

Dramiński M, Rada-Iglesias A, Enroth S, Wadelius C, Koronacki J, Komorowski HJ (2008) Monte carlo feature selection for supervised classification. Bioinformatics 24(1):110–117

Dramiński M, Kierczak M, Nowak-Brzezińska A, Koronacki J, Komorowski J (2011) The Monte Carlo feature selection and interdependency discovery is unbiased. Control Cybernet 40(2):199–211

Draper B, Brodley CE, Utgoff PE (1994) Goal-directed classification using linear machine decision trees. IEEE Trans Pattern Anal Mach Intell 16:888–893

Duch W, Biesiada J, Winiarski T, Grudziński K, Grąbczewski K (2002) Feature selection based on information theory filters. In: Proceedings of the international conference on neural networks and soft computing (ICNNSC 2002) Physica-Verlag (Springer). Zakopane, Advances in Soft Computing, pp 173–176

Duch W, Grąbczewski K (2002) Heterogeneous adaptive systems. In: Proceedings of the world congress of computational intelligence, Honolulu

Duch W, Winiarski T, Biesiada J, Kachel A (2003) Feature selection and ranking filters. In: Artificial neural networks and neural information processing - ICANN/ICONIP 2003, Istanbul, pp 251–254

Duda RO, Hart PE, Stork DG (2001) Pattern classification, 2nd edn. John Wiley and Sons, New York

Efron B (1983) Estimating the error rate of a prediction rule: Improvement on cross-validation. J Am Stat Assoc 78(382):316–331. http://www.jstor.org/stable/2288636

Efron B, Tibshirani R (1997) Improvements on cross-validation: The.632+ bootstrap method. J Am Stat Assoc 92(438):548–560. http://www.jstor.org/stable/2965703

Esposito F, Malerba D, Semeraro G (1997) A comparative analysis of methods for pruning decision trees. IEEE Trans Pattern Anal Mach Intell 19(5):476–491

Fayyad UM, Irani KB (1992a) The attribute selection problem in decision tree generation. In: Proceedings of the tenth national conference on artificial intelligence, AAAI'92. AAAI Press, pp 104–110

Fayyad UM, Irani KB (1992b) On the handling of continuous-valued attributes in decision tree generation. Mach Learn 8:87–102. doi:10.1007/BF00994007

Ferri C, Flach PA, Hernández-Orallo J (2002) Learning decision trees using the area under the ROC curve. In: ICML '02: Proceedings of the nineteenth international conference on machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 139–146

Ferri C, Flach P, Hernández-Orallo J (2003) Improving the auc of probabilistic estimation trees. In: Lavrac N, Gamberger D, Blockeel H, Todorovski L (eds) Machine learning: ECML 2003, Lecture Notes in Computer Science, vol 2837. Springer, Berlin, pp 121–132. doi:10.1007/978-3-540-39857-8_13

Fierens D, Ramon J, Blockeel H, Bruynooghe M (2005) A comparison of approaches for learning probability trees. In: Gama J, Camacho R, Brazdil P, Jorge AM, Torgo L (eds) Machine learning: ECML 2005. Lecture Notes in Computer Science, vol 3720. Springer, Berlin, pp 556–563. doi:10.1007/11564096_54

Fournier D, Crémilleux B (2002) A quality index for decision tree pruning. Knowl-Based Syst 15(1–2):37–43

Frank E, Witten IH (1998) Using a permutation test for attribute selection in decision trees. In: International conference on machine learning. Morgan Kaufmann, pp 152–160

Frean MR (1990) Small nets and short paths: optimising neural computation. PhD dissertation, University of Edinburgh

Freund Y, Schapire R (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139

Freund Y, Mason L (1999) The alternating decision tree learning algorithm. In: Proceedings of ICML 99, Bled, Slovenia, pp 124–133

Freund Y, Schapire R (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In: Vitanyi P (ed) Computational learning theory. Lecture Notes in Computer Science, vol 904. Springer, Berlin, pp 23–37. doi:10.1007/3-540-59119-2_166

Freund Y, Schapire R (1996) Experiments with a new boosting algorithm. In: Proceedings of the thirteenth international conference on machine learning, pp 148–156

Friedman JH (1999a) Greedy function approximation: a gradient boosting machine. Tech. rep., Department of Statistics, Stanford University

Friedman JH (1999b) Stochastic gradient boosting. Comput Stat Data Anal 38:367–378

Friedman JH (1977) A recursive partitioning decision rule for nonparametric classification. IEEE Trans Comput 100(4):404–408

Gama J (1997) Probabilistic linear tree. In: ICML '97: Proceedings of the fourteenth international conference on machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 134–142

Gama J (1999) Discriminant trees. In: ICML '99: Proceedings of the sixteenth international conference on machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 134–142

Garofalakis M, Hyun D, Rastogi R, Shim K (2000) Efficient algorithms for constructing decision trees with constraints. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, pp 335–339

Garofalakis M, Hyun D, Rastogi R, Shim K (2003) Building decision trees with constraints. Data Mining Knowl Discov 7:187–214. doi:10.1023/A:1022445500761

Gehrke J, Ganti V, Ramakrishnan R, Loh WY (1999) BOAT - optimistic decision tree construction

Gehrke J, Ramakrishnan R, Ganti V (1998) Rainforest: a framework for fast decision tree construction of large datasets. In: VLDB. Morgan Kaufmann, pp 416–427

Gehrke J, Ramakrishnan R, Ganti V (2000) Rainforest-a framework for fast decision tree construction of large datasets. Data Mining Knowl Discov 4:127–162. doi:10.1023/A:1009839829793

Gnanadesikan R (1977) Methods for statistical data analysis of multivariate observations. John Wiley, New York

Good I (1965) The estimation of probabilities. MIT Press, Cambridge

Good P (1994) Permutation tests. Springer, New York

Goodman RM, Smyth P (1988a) Decision tree design from a communication theory standpoint. IEEE Trans Inform Theory 34(5):979–994. doi:10.1109/18.21221

Goodman RM, Smyth P (1988b) An information theoretic model for rule-based expert systems. In: International symposium on information theory, Kobe, Japan

Goodman RM, Smyth P (1988c) Information theoretic rule induction. In: Proceedings of the 1988 conference on AI. Pitman Publishing, London

Grąbczewski K (2003) Zastosowanie kryterium separowalności do generowania regułklasyfikacji na podstawie baz danych. PhD thesis, Systems Research Institute, Polish Academy of Sciences, Warsaw

Grąbczewski K (2004) SSV criterion based discretization for Naive Bayes classifiers. In: Proceedings of the 7th international conference on artificial intelligence and soft computing, Zakopane, Poland

Grąbczewski K (2011) Separability of split value criterion with weighted separation gains. In: Perner P (ed) Machine learning and data mining in pattern recognition, Lecture Notes in Computer Science, vol 6871. Springer, Berlin, pp 88–98. doi:10.1007/978-3-642-23199-5_7

Grąbczewski K (2012) Decision tree cross-validation committees. Data Mining Knowl Discov, submitted. http://www.is.umk.pl/kg/papers/12-DTCVComm.pdf

Grąbczewski K, Duch W (1999) A general purpose separability criterion for classification systems. In: Proceedings of the 4th conference on neural networks and their applications, Zakopane, Poland, pp 203–208

Grąbczewski K, Duch W (2000) The separability of split value criterion. In: Proceedings of the 5th conference on neural networks and their applications, Zakopane, Poland, pp 201–208

Grąbczewski K, Duch W (2002a) Forests of decision trees. In: Proceedings of international conference on neural networks and soft computing, Physica-Verlag (Springer), Advances in Soft Computing, pp 602–607

Grąbczewski K, Duch W (2002b) Heterogeneous forests of decision trees. In: Proceedings of international conference on artificial neural networks. Lecture Notes in Computer Science, vol 2415. Springer, pp 504–509

Grąbczewski K, Jankowski N (2005) Feature selection with decision tree criterion. In: Nedjah N, Mourelle L, Vellasco M, Abraham A, Köppen M (eds) Fifth international conference on hybrid intelligent systems. IEEE, Computer Society, Rio de Janeiro, Brazil, pp 212–217

Grąbczewski K, Jankowski N (2006) Mining for complex models comprising feature selection and classification. In: Guyon I, Gunn S, Nikravesh M, Zadeh L (eds) Feature extraction, foundations and applications. Springer, Berlin, pp 473–489

Green DM, Swets JA (1966) Signal detection theory and psychophysics. John Wiley, New York

Guo H, Gelfand SB (1992) Classification trees with neural network feature extraction. IEEE Trans Neural Netw 3(6):923–933

Hand DJ, Till RJ (2001) A simple generalisation of the area under the ROC curve for multiple class classification problems. Mach Learn 45(2):171–186. doi:10.1023/A:1010920819831

Hartigan JA, Wong MA (1979) Algorithm as 136: a k-means clustering algorithm. J R Stat Soc Ser C (Appl Stat) 28(1):100–108

Hastie T, Tibshirani R (1990) Generalized additive models. Chapman and Hall, London

Hawkins DM (1999) FIRM: formal inference-based recursive modeling. Tech. Rep. 546, School of Statistics, University of Minnesota

Heath D, Kasif S, Salzberg S (1993) Induction of oblique decision trees. J Artif Intell Res 2(2):1–32

Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learn 11:63–91

Hothorn T, Hornik K, Wiel MAVD, Zeileis A (2006a) A lego system for conditional inference. Am Stat 60:257–263

Hothorn T, Hornik K, Zeileis A (2006b) Unbiased recursive partitioning: a conditional inference framework. J Comput Graph Stat 15(3):651–674

Hothorn T, Hornik K, van de Wiel MA, Zeileis A (2008) Implementing a class of permutation tests: the coin package. J Stat Softw 28(8):1–23. http://www.jstatsoft.org/v28/i08

Hothorn T, Hornik K, Zeileis A (2004) Unbiased recursive partitioning: a conditional inference framework. Research Report Series 8, Department of Statistics and Mathematics, Institut für Statistik und Mathematik, WU Vienna University of Economics and Business, Vienna

Huang Z, Ng MK, Lin T, Cheung DWL (2000) An interactive approach to building classification models by clustering and cluster validation. In: Proceedings of the second international conference on intelligent data engineering and automated learning, data mining, financial engineering, and intelligent agents. Springer, London, UK, IDEAL '00, pp 23–28. http://dl.acm.org/citation.cfm?id=646287.688767

Huber PJ (1977) Robust statistical procedures. Society for Industrial and Applied Mathematics, Pittsburgh

Hubert L, Arabie P (1985) Comparing partitions. J Classif 2:193–218. doi:10.1007/BF01908075

Janssen F, Fürnkranz J (2007) On meta-learning rule learning heuristics. In: ICDM, pp 529–534

Janssen F, Fürnkranz J (2008) An empirical comparison of hill-climbing and exhaustive search in inductive rule learning

Janssen F, Fürnkranz J (2008) An empirical investigation of the trade-off between consistency and coverage in rule learning heuristics. In: Discovery Science, pp 40–51

Janssen F, Fürnkranz J (2009) A re-evaluation of the over-searching phenomenon in inductive rule learning. In: Proceedings of the SIAM international conference on data mining (SDM-09), pp 329–340

Janssen F, Fürnkranz J (2010)On the quest for optimal rule learning heuristics. Mach Learn 78:343–379. doi:10.1007/s10994-009-5162-2

Jenhani I, Amor NB, Elouedi Z (2008) Decision trees as possibilistic classifiers. Int J Approx Reason 48(3):784–807. doi:10.1016/j.ijar.2007.12.002

John GH (1995a) Robust decision trees: removing outliers in databases. In: First international conference on knowledge discovery and data mining. AAAI Press, Menlo Park, CA, pp 174–179

John GH (1995b) Robust linear discriminant trees. In: AI&Statistics-95 [7]. Springer-Verlag, pp 285–291

John GH (1996) Robust linear discriminant trees. In: Fisher D, Lenz H (eds) Learning from data: artificial intelligence and statistics V. Lecture Notes in Statistics, Springer-Verlag, New York, chap 36:375–385

Kass GV (1980) An exploratory technique for investigating large quantities of categorical data. Appl Stat 29:119–127

Kearns M, Mansour Y (1999) On the boosting ability of top-down decision tree learning algorithms. J Comput Syst Sci 58(1):109–128

Kim JH (2009) Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. Comput Stat Data Anal 53(11):3735–3745. http://ideas.repec.org/a/eee/csdana/v53y2009i11p3735-3745.html

Kim H, Loh WY (2001) Classification trees with unbiased multiway splits. J Am Stat Assoc 96:589–604. http://www.stat.wisc.edu/loh/treeprogs/cruise/cruise.pdf

Kim H, Loh WY (2003) Classification trees with bivariate linear discriminant node models. J Comput Graph Stat 12:512–530. http://www.stat.wisc.edu/loh/treeprogs/cruise/jcgs.pdf

Kira K, Rendell LA (1992a) The feature selection problem: traditional methods and a new algorithm. In: Proceedings of the national conference on artificial intelligence. John Wiley & Sons Ltd, pp 129–134

Kira K, Rendell LA (1992b) A practical approach to feature selection. In: ML92: Proceedings of the ninth international workshop on machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 249–256

Kohavi R, Kunz C (1997) Option decision trees with majority votes. In: Proceedings of the fourteenth international conference on machine learning, pp 161–169

Kohavi R, Sommerfield D, Dougherty J (1996) Data mining using MLC++: a machine learning library in C++. In: Tools with artificial intelligence. IEEE Computer Society Press, pp 234–245. http://www.sgi.com/tech/mlc

Kononenko I (1994) Estimating attributes: analysis and extensions of RELIEF. In: European conference on machine learning. Springer, pp 171–182

Kononenko I (1995) On biases in estimating multi-valued attributes. In: Proceedings of the 14th international joint conference on artificial intelligence, IJCAI'95, vol 2. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1034–1040. http://dl.acm.org/citation.cfm?id=1643031.1643034

Kononenko I (1998) The minimum description length based decision tree pruning. In: Lee HY, Motoda H (eds) PRICAI'98: topics in artificial intelligence. Lecture Notes in Computer Science, vol 1531. Springer Berlin, pp 228–237

Kotsiantis S (2011) Decision trees: a recent overview. Artif Intell Rev 35:1–23. doi:10.1007/s10462-011-9272-4

Koziol JA (1991) On maximally selected chi-square statistics. Biometrics 47(4):1557–1561. URL http://www.jstor.org/stable/2532406

Kuncheva LI, Whitaker CJ (2002) Using diversity with three variants of boosting: aggressive, conservative, and inverse. In: Roli F, Kittler J (eds) Multiple classifier systems. Lecture Notes in Computer Science, vol 2364. Springer, Berlin, pp 81–90. doi:10.1007/3-540-45428_48

Lee JY, Olafsson S (2006) Multi-attribute decision trees and decision rules. In: Triantaphyllou E, Felici G (eds) Data mining and knowledge discovery approaches based on rule induction techniques, massive computing, vol 6. Springer US, pp 327–358. Doi:10.1007/0-387-34296-6_10

Lee TH, Shih YS (2006) Unbiased variable selection for classification trees with multivariate responses. Comput Stat Data Anal 51(2):659–667

Levene H (1960) Robust tests for equality of variances. In: Olkin I (ed) Contributions to probability and statistics. Stanford University Press, Palo Alto, pp 278–292

Li Y, Hung E, Chung K, Huang J (2008) Building a decision cluster classification model for high dimensional data by a variable weighting k-means method. In: Proceedings of the twenty-first Australasian joint conference on artificial intelligence, Auckland, pp 337–347

Lim TS, Loh WY, Shih YS (2000) A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. Mach Learn 40:203–228

Liu W, Chawla S, Cieslak DA, Chawla NV (2010) A robust decision tree algorithm for imbalanced data sets. In: SDM, SIAM, pp 766–777. http://dblp.uni-trier.de/rec/bibtex/conf/sdm/LiuCCC10

Loh WY (2002) Regression trees with unbiased variable selection and interaction detection. Stati Sin 12:361–386. http://www3.stat.sinica.edu.tw/statistica/j12n2/j12n21/j12n21.htm

Loh WY, Vanichsetakul N (1988) Tree-structured classification via generalized discriminant analysis (with discussion). J Am Stat Assoc 83:715–728

Loh WY, Shih YS (1997) Split selection methods for classification trees. Stat Sin 7:815–840

Magidson J (1993) The use of the new ordinal algorithm in chaid to target profitable segments. J Database Market 1:29–48

Malerba D, Esposito F, Semeraro G (1996) A further comparison of simplification methods for decision-tree induction. In: In Fisher D, Lenz H (eds) Learning. Springer, Berlin, pp 365–374

Mballo C, Diday E (2006) The criterion of kolmogorov-smirnov for binary decision tree: application to interval valued variables. Intell Data Anal 10(4):325–341

Mehta M, Agrawal R, Rissanen J (1996) SLIQ: a fast scalable classifier for data mining. In: Proceedings of the 5th international conference on extending database technology: advances in database technology. Springer, London, UK, EDBT '96, pp 18–32. URL http://dl.acm.org/citation.cfm?id=645337.650384

Mehta M, Rissanen J, Agraval R (1995) MDL-based decision tree pruning. In: Fayyad U, Uthurusamy R (eds) Proceedings of the first international conference on knowledge discovery and data mining. AAAI Press, Menlo Park, CA, pp 216–221

Melville P, Mooney RJ (2003) Constructing diverse classifier ensembles using artificial training examples. In: Proceedings of the eighteenth international joint conference on artificial intelligence, pp 505–510

Michie D (1990) Personal models of rationality. J Stat Plan Infer 25(3):381–399. http://www.sciencedirect.com/science/article/B6V0M-45SJDGS-F/1/17548ffdb8fe70dfd840185272bdbcdf

Michie D, Spiegelhalter DJ, Taylor CC (1994) Machine learning, neural and statistical classification. Elis Horwood, London

Mierswa I, Wurst M, Klinkenberg R, Scholz M, Euler T (2006) Yale: Rapid prototyping for complex data mining tasks. In: Ungar L, Craven M, Gunopulos D, Eliassi-Rad T (eds) Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '06. ACM, New York, NY, USA, pp 935–940. URL http://rapid-i.com/component/option,com_docman/task,doc_download/gid,25/Itemid,62

Mingers J (1989a) An empirical comparison of pruning methods for decision tree induction. Mach Learn 4(2):227–243

Mingers J (1989b) An empirical comparison of selection measures for decision-tree induction. Mach Learn 3:319–342

Mitchell T (1997) Machine learning. McGraw Hill, New York

Morgan JN, Sonquist JA (1963a) Problems in the analysis of survey data, and a proposal. J Am Stat Assoc 58(302):415–434. http://www.jstor.org/stable/2283276

Morgan JN, Sonquist JA (1963b) Some results from a non-symmetrical branching process that looks for interaction effects. In: Proceedings of the social statistics section. American Statistical Association, pp 40–53

Müller W, Wysotzki F (1994) Automatic construction of decision trees for classification. Ann Oper Res 52:231–247

Müller W, Wysotzki F (1997) The decision-tree algorithm CAL5 based on a statistical approach to its splitting algorithm. Machine learning and statistics: the interface, pp 45–65

Murthy SK (1997) On growing better decision trees from data. PhD thesis, The Johns Hopkins University, Baltimore, MD

Murthy SK (1998) Automatic construction of decision trees from data: a multi-disciplinary survey. Data Mining Knowl Discov 2:345–389. doi:10.1023/A:1009744630224

Murthy SK, Salzberg S (1995) Lookahead and pathology in decision tree induction. In: Proceedings of the 14th international joint conference on artificial intelligence. Morgan Kaufmann, pp 1025–1031

Murthy SK, Kasif S, Salzberg S (1994) A system for induction of oblique decision trees. J Artif Intell Res 2:1–32

Murthy S, Kasif S, Salzberg S, Beigel R (1993) Oc1: randomized induction of oblique decision trees. In: AAAI'93, pp 322–327

Nettleton D, Banerjee T (2001) Testing the equality of distributions of random vectors with categorical components. Comput Stat Data Anal 37(2):195–208. http://www.sciencedirect.com/science/article/pii/S0167947301000159

Niblett T (1989) Functional specification for realid. Tech. rep.

Niblett T, Bratko I (1986) Learning decision rules in noisy domains. In: Proceedings of expert systems '86, the 6th annual technical conference on research and development in expert systems III. Cambridge University Press, New York, NY, USA, pp 25–34

Nijssen S, Fromont E (2010) Optimal constraint-based decision tree induction from itemset lattices. Data Mining Knowl Discov 21:9–51. doi:10.1007/s10618-010-0174-x

Noh HG, Song MS, Park SH (2004) An unbiased method for constructing multilabel classification trees. Comput Stat Data Anal 47(1):149–164. http://www.sciencedirect.com/science/article/pii/S0167947303002433

Oates T, Jensen D (1999) Toward a theoretical understanding of why and when decision tree pruning algorithms fail. In: Proceedings of the sixteenth national conference on artificial intelligence and the eleventh innovative applications of artificial intelligence conference innovative applications of artificial intelligence. American Association for Artificial Intelligence, Menlo Park, CA, USA, AAAI '99/IAAI '99, pp 372–378. http://dl.acm.org/citation.cfm?id=315149.315327

O'Keefe RA (1983) Concept formation from very large training sets. In: Proceedings of the eighth international joint conference on Artificial intelligence, vol 1. Morgan Kaufmann, San Francisco, CA, USA, IJCAI'83. pp 479–481, http://dl.acm.org/citation.cfm?id=1623373.1623490

Oliveira A, Sangiovanni-Vincentelli A, Shavlik J (1996) Using the minimum description length principle to infer reduced ordered decision graphs. In: Machine Learning, pp 23–50

Oza NC (2003) Boosting with averaged weight vectors. In: Proceedings of the 4th international conference on multiple classifier systems. Springer, Berlin, MCS'03, pp 15–24. http://dl.acm.org/citation.cfm?id=1764295.1764299

Parmanto B, Munro PW, Doyle HR (1995) Improving committee diagnosis with resampling techniques. In: NIPS, pp 882–888

Piccarreta R (2008) Classification trees for ordinal variables. Comput. Stat. 23:407–427. doi:10.1007/s00180-007-0077-5

Provost F, Kolluri V (1999) A survey of methods for scaling up inductive algorithms. Data Mining Knowl Discov 3:131–169. doi:10.1023/A:1009876119989

Quinlan JR, Cameron-Jones RM (1995) Oversearching and layered search in empirical learning. In: IJCAI, pp 1019–1024

Quinlan JR (1987) Simplifying decision trees. Int J Man-Mach Stud 27(3):221–234. doi:10.1016/S0020-7373(87)80053-6

Quinlan JR (1993) C 4.5: programs for machine learning. Morgan Kaufmann, San Mateo

Quinlan JR (1996) Bagging, boosting, and C4.5. In: Proceedings of the thirteenth national conference on artificial intelligence and eighth innovative applications of artificial intelligence conference, AAAI 96, IAAI 96, vol 1. AAAI Press/The MIT Press, Portland, Oregon, pp 725–730

Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106

Quinlan JR, Rivest RL (1989) Inferring decision trees using the minimum description length principle. Inform Comput 80(3):227–248

Rastogi R, Shim K (2000) Public: a decision tree classifier that integrates building and pruning. Data Mining Knowl Discov 4:315–344. doi:10.1023/A:1009887311454

Rokach L, Maimon O (2008) Data mining with decision trees: theory and applications. World Scientific, Singapore

Rokach L, Maimon O (2010) Classification trees. In: Maimon O, Rokach L (eds) Data mining and knowledge discovery handbook. Springer US, pp 149–174. doi:10.1007/978-0-387-09823-4-9

Rutkowski L, Pietruczuk L, Duda P, Jaworski M (2012) Decision trees for mining data streams based on the McDiarmid's bound. IEEE Trans Knowl Data Eng PP(99):1–14

Rzepakowski P, Jaroszewicz S (2012) Decision trees for uplift modeling with single and multiple treatments. Knowl Inform Syst 32(2):303–327

Safavian S, Landgrebe D (1991) A survey of decision tree classifier methodology. IEEE Trans Syst Man Cybernet 21(3):660–674. doi:10.1109/21.97458

Schapire RE, Singer Y (1999) Improved boosting algorithms using confidence-rated predictions. Mach Learn 37(3):297–336

Schlimmer JC, Fisher D (1986) A case study of incremental concept induction. In: Proceedings of the fifth national conference on artificial intelligence. Morgan Kaufmann, Philadelphia, PA, pp 496–501

Seewald AK, Petrak J, Widmer G (2000) Hybrid decision tree learners with alternative leaf classifiers: an empirical study. In: Proceedings of the 14th FLAIRS conference. AAAI Press, pp 407–411

Segal R (1996) An analysis of oversearch. Unpublished manuscript

Shafer JC, Agrawal R, Mehta M (1996) SPRINT: a scalable parallel classifier for data mining. In: Proceedings of the 22nd international conference on very large data bases. Morgan Kaufmann

Publishers Inc., San Francisco, CA, USA, VLDB '96, pp 544–555. http://dl.acm.org/citation.cfm?id=645922.673491

Shih YS (1999) Families of splitting criteria for classification trees. Stat Comput 9:309–315. doi:10.1023/A:1008920224518

Shih YS (2004) A note on split selection bias in classification trees. Comput Stat Data Anal 45:457–466

Siciliano R, Mola F (2000) Multivariate data analysis and modeling through classification and regression trees. Comput Stat Data Anal 32(3–4):285–301. http://www.sciencedirect.com/science/article/pii/S0167947399000821

Smyth P, Goodman RM (1992) An information theoretic approach to rule induction from databases. IEEE Trans Knowl Data Eng 4(4):301–316

Snedecor GW, Cochran W (1989) Statistical Methods. No. 276 in Statistical Methods, Iowa State University Press

Srivastava A, Han EH, Kumar V, Singh V (1999) Parallel formulations of decision-tree classification algorithms. Data Mining Knowl Discov 3:237–261. doi:10.1023/A:1009832825273

Strasser H, Weber C (1999) On the asymptotic theory of permutation statistics. Math Meth Stat 2:220–250

Strobl C, Boulesteix AL, Augustin T (2005) Unbiased split selection for classification trees based on the gini index. Ludwig-Maximilian University, Munich, Tech. rep.

Tadeusiewicz R, Izworski A, Majewski J (1993) Biometria. Wydawnictwa AGH, Kraków

Therneau TM, Atkinson EJ (1997) An introduction to recursive partitioning using the RPART routines. Tech. rep., Divsion of Biostatistics 61, Mayo Clinic

Torres-Sospedra J, Hernández-Espinosa C, Fernández-Redondo M (2007) Averaged conservative boosting: introducing a new method to build ensembles of neural networks. In: de Sá J, Alexandre L, Duch W, Mandic D (eds) Artificial neural networks - ICANN 2007. Lecture Notes in Computer Science, vol 4668. Springer, Berlin, pp 309–318

Utgoff PE (1989) Incremental induction of decision trees. Mach Learn 4:161–186. Doi:10.1023/A:1022699900025

Utgoff PE (1994) An improved algorithm for incremental induction of decision trees. In: Proceedings of the eleventh international conference on machine learning. Morgan Kaufmann, pp 318–325

Utgoff PE, Brodley CE (1991) Linear machine decision trees. Tech. Rep. UM-CS-1991-010, Department of Computer Science, University of Massachusetts

Utgoff PE, Clouse JA (1996) A Kolmogorov-Smirnoff metric for decision tree induction. Tech. rep., University of Massachusetts, Amherst, MA, USA

Vinh NX, Epps J, Bailey J (2010) Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance. J Mach Learn Res 11:2837–2854. http://dl.acm.org/citation.cfm?id=1756006.1953024

Voisine N, Boullé M, Hue C (2009) A bayes evaluation criterion for decision trees. In: Advances in knowledge discovery and management (AKDM09)

Wallace C, Patrick J (1993) Coding decision trees. Mach Learn 11:7–22. doi:10.1023/A:1022646101185

Wang H, Zaniolo C (2000) CMP: a fast decision tree classifier using multivariate predictions. In: Proceedings of the 16th international conference on data engineering, pp 449–460

White AP, Liu WZ (1994) Bias in information-based measures in decision tree induction. Mach Learn 15:321–329. doi:10.1023/A:1022694010754

Wilson EB (1927) Probable inference, the law of succession, and statistical inference. J Am Stat Assoc 22:209–212

Wozniak M (2011) A hybrid decision tree training method using data streams. Knowl Inform Syst 29:335–347. doi:10.1007/s10115-010-0345-5

Yildiz OT, Alpaydin E (2000) Linear discriminant trees. In: ICML '00: Proceedings of the seventeenth international conference on machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1175–1182

Yildiz OT, Alpaydin E (2005a) Linear discriminant trees. Int J Pattern Recogn Artif Intell 19(3):323–353

Yildiz OT, Alpaydin E (2005b) Model selection in omnivariate decision trees. In: 16th European conference on machine learning, Porto, Portugal, pp 473–484

Yildiz OT, Alpaydin E (2001) Omnivariate decision trees. IEEE Trans Neural Netw 12(6):1539–1546

Yildiz OT (2011) Model selection in omnivariate decision trees using structural risk minimization. Inform Sci 181:5214–5226

Zeileis A, Hothorn T, Hornik K (2008) Model-based recursive partitioning. J Comput Graph Stat 17(2):492–514. http://statmath.wu.ac.at/zeileis/papers/Zeileis+Hothorn+Hornik-2008.pdf

Zenobi G, Cunningham P (2001) Using diversity in preparing ensembles of classifiers based on different feature seature subsets to minimize generalization error. In: Lecture Notes in Computer Science. Springer, pp 576–587

Zhang H (1998) Classification trees for multiple binary responses. J Am Stat Assoc 93(441):180–193. http://www.jstor.org/stable/2669615

Zhang H, Su J (2006) Learning probabilistic decision trees for auc. Pattern Recogn Lett 27(8):892–899. ROC Analysis in Pattern Recognition. http://www.sciencedirect.com/science/article/pii/S0167865505003065

Springer

http://www.springer.com/978-3-319-00959-9

Meta-Learning in Decision Tree Induction
Grąbczewski, K.
2014, XVI, 343 p. 33 illus., Hardcover
ISBN: 978-3-319-00959-9