Comput Stat
DOI 10.1007/s00180-011-0242-8

ORIGINAL PAPER

Reusable components in decision tree induction algorithms

Milija Suknovic · Boris Delibasic · Milos Jovanovic · Milan Vukicevic · Dragana Becejski-Vujaklija · Zoran Obradovic

Received: 18 February 2009 / Accepted: 5 February 2011

Abstract We propose a generic decision tree framework that supports reusable components design. The proposed generic decision tree framework consists of several sub-problems which were recognized by analyzing well-known decision tree induction algorithms, namely ID3, C4.5, CART, CHAID, QUEST, GUIDE, CRUISE, and CTREE. We identified reusable components in these algorithms as well as in several of their partial improvements that can be used as solutions for sub-problems in the generic decision tree framework. The identified components can now be used outside the algorithm they originate from. Combining reusable components allows the replication of original algorithms, their modification but also the creation of new decision tree induction algorithms. Every original algorithm can outperform other algorithms under specific conditions but can also perform poorly when these conditions change. Reusable components allow exchanging of solutions from various algorithms and fast design of new algorithms. We offer a generic framework for component-based algorithms design that enhances understanding, testing and usability of decision tree algorithm parts.

Keywords Decision tree · Algorithm · Framework · Reusable components · Generic · Design

M. Suknovic · B. Delibasic (✉) · M. Jovanovic · M. Vukicevic · D. Becejski-Vujaklija
Faculty of Organizational Sciences, University of Belgrade,
Jove Ilica 154, Belgrade, Serbia
e-mail: boris.delibasic@fon.bg.ac.rs

Z. Obradovic
Information Science and Technology Center, Temple University, Philadelphia, PA 19122, USA

1 Introduction

The research presented in this paper deals with design of decision tree algorithms based on reusable components (RCs) and their identification in some popular algorithms and some of their partial improvements. RCs are solutions for typical sub-problems in design. Several decision tree algorithms and their partial improvements were analyzed throughout this paper. We analyzed decision tree algorithms ID3 (Quinlan 1986), C4.5 (Quinlan 1993), CART (Breiman et al. 1984), CHAID (Kass 1980), QUEST (Loh and Shih 1997), CRUISE (Kim and Loh 2001), GUIDE (Loh 2002), and CTREE (Hothorn et al. 2006).

The choice of these algorithms was based on availability in popular open source software (e.g. (R Development Core Team 2008), Weka (Witten and Frank 2005), RapidMiner (Mierswa et al. 2006)), the number of citations (e.g. on Google Scholar algorithms and corresponding number of citations for analyzed algorithms are: C4.5 15327, CART 14434, ID3 8599, CHAID 909, QUEST 420, CRUISE 142, GUIDE 131, CTREE 115) as well as design issues the algorithms propose. We also analyzed some partial improvements of these algorithms (e.g. Mantaras 1991). Finally, we identified RCs that can be used to construct inductive decision tree algorithms.

We propose a generic decision tree algorithm, with a structure consisting of sub-problems with predefined inputs and outputs. To build a decision tree algorithm several sub-problems should be solved. RCs with the same input-output structure can be applied to resolve each of the sub-problems. Some sub-problems are crucial for an algorithm, and some are optional.

The need for reusable components that can be interchanged between algorithms is emphasized in Sonnenburg et al. (2007). Our research supports the call for standardization, exchange and fair testing of algorithms and their parts as suggested in Sonnenburg et al. (2007). The aforementioned study reports that benefits from open source frameworks design include:

- Combining advantages of various algorithms,
- Easier reproduction of scientific results,
- Comparing algorithms in more details,
- Building on existing resources with less re-implementation,
- Faster adaptation in other disciplines and industry,
- Collaborative emergence of standards.

The paper is structured as follows. The related work in this area of research is summarized in Sect. 2. Section 3 contains description and brief analysis of the decision tree algorithms we considered. The process of identifying RCs and description of RCs considered in this study are contained in Sect. 4. In Sect. 5 the generic decision tree framework is proposed followed by discussion and further research directions contained in Sect. 6.

2 Related work

Some comparison of decision tree algorithm designs can be found in Murthy (1998), Safavian and Landgrebe (1991). Key topics of decision tree design are discussed in these papers. Combining of advantages between decision tree algorithms is, however, mostly done with hybrid algorithms. There are many hybrid decision tree algorithms in the literature that combine various machine learning algorithms (e.g. Siddique et al. 2007).

Combining of whole, already implemented decision tree components is, on the other hand, found rare in the literature. In Delibasic et al. (2010) experimental evidences of advantages of generic tree design are presented. A generic decision tree is also proposed in Hothorn et al. (2006), Zeileis et al. (2008). Hothorn states that the separation of variable selection and splitting procedure in the algorithm is "the key for the construction of interpretable tree structures not suffering a systematic tendency towards covariates with many possible splits or many missing values" (Hothorn et al. 2006, p 4). Another research in decision-tree design can be also found in Drossos et al. (2000).

We believe that most authors of decision tree classifiers are well aware of the nature of design. Quinlan's C4.5 (Quinlan 1993) presents not only a more efficient algorithm compared to ID3 (Quinlan 1986), but also has some design improvements, that will be discussed further in this paper.

An interesting development of decision tree algorithms is conducted under Loh et al. In their algorithms FACT (Loh and Vanichsetakul 1988), QUEST (Loh and Shih 1997), CRUISE (Kim and Loh 2001) many statistical and machine learning components are tied together. GUIDE (Loh 2002) is a regression tree learner that uses well established ideas for building classification trees. Most of the work done by now suggests incremental improvement of algorithms. Our research supports that way of design improvements, but allows even more.

Many authors discuss reusable-components design. According to Freeman (1983), reusable components are usually source code fragments, but they can also include design structures, module-level implementation structures, specifications, documentation or transformations. Sommerville (2004) identifies several widely applied design reuse approaches, e.g. design pattern, application frameworks or program libraries. Moreover, reuse is possible at a range of levels from simple functions to complete systems and from more abstract descriptions to concrete applications.

Another popular theory that deals with reusable components design is the pattern theory. Originally it was introduced by architect Christopher Alexander (1979). Since then, it has been successfully applied to software engineering (Coplien and Schmidt 1995; Gamma et al. 1995), organizational design (Coplien and Harrison 2005), telecommunication design (Adams et al. 1991), avionics control system design (Lea 1994). The pattern approach has earned its popularity over the years and is being accepted in many areas of human creative activity. There are also first papers discussing RCs in data mining (e.g. Delibasic et al. 2008, 2009).

Winn and Calder (2002) proposed an open list of reusable components features that can be used for RC identification. But even with the help of this list, it is difficult to decide whether a component is a RC or not because the process of identifying reusable components is not formalized.

For the description of RCs, Tracz's definition (Tracz 1990) is helpful. He defines RCs as triplets consisting of concept, content and context. The concept is the description of what a component does; it describes the interface and the semantics represented by pre and post conditions. The content describes how the component is realized, which is hidden from the casual user. The context describes the application domain of the component, which helps to find the appropriate component for a specific problem. Every reusable component is well-documented, and describes where the component can be applied, but it also includes a solution for the problem it solves. A reusable component is abstract in a sense that it can have many areas of usage, but it is also specific because it offers a concrete solution.

3 Decision tree based classifiers

The following notations will be used throughout this paper. A dataset S consist of n cases with m input attributes X and one output (dependent) attribute Y (numerical or categorical). The goal of decision tree algorithms is to find a decision tree model that, based on the values of X, classifies cases into Y classes (classification model), or estimates the value of Y (regression model). A decision tree model consists of nodes, branches, and leaves. Leaves define the prediction of Y, and a rule that can be obtained by descending the tree from the root to the leaf. The number of leaves defines the number of rules that can be extracted from a decision tree. Trees are suitable for obtaining IF-THEN rules in a hierarchical tree-like structure.

Decision trees can be grown in various ways. Our paper analyzes how C4.5-like algorithms grow trees (ID3, CART, CHAID). These algorithms generally follow three steps:

1. Generate for a given dataset S possible splitting candidates, i.e. for all attributes $X_i (i \in 1, \ldots, m)$ define possible splits (potential nodes branches which are obtained by defining possible cut points for numerical attributes, and possible groupings of categories for categorical attributes).
2. Evaluate the generated splits with a split evaluation measure and choose the best split. Implement the split, i.e. grow the tree with the chosen split as a node in the tree. The dataset S is then divided into disjoint subsets $S_j (j \in 1, \ldots, k)$, where k defines number of branches of the chosen split, having $S = \cap S_j$.
3. Repeat steps 1 and 2 recursively for all branches $\left(S_j\right)$ of the tree until tree is grown completely (i.e. all leaves of the decision tree are "pure") or until another user-defined stopping criteria has been fulfilled (e.g. maximal tree depth, node significance threshold, etc.).

After the tree model has been grown, pruning of trees can be performed additionally to resolve the decision tree model's overfitting problem.

We stress out that these steps are not applicable for all decision tree algorithms, since there are many decision tree algorithms that don't follow these steps. E.g. FACT-like algorithms (e.g. QUEST, CRUISE, GUIDE, CTREE) have different algorithm steps:

1. Find the most appropriate attribute $Xi (i \in 1, \ldots, m)$ for splitting with respect to Y. This attribute will present the node of a tree.
2. Find the attribute's most appropriate split with respect to Y. The split will form branches of the node.

3. Repeat steps 1 and 2 until tree is grown completely or until another user-defined stopping criteria has been reached.

We have only analyzed parts of the algorithms related to the growth and pruning phase. A short overview of the analyzed parts of the algorithms is further presented.

3.1 ID3

The "Iterative Dichotomiser Tree" (ID3) works only with categorical input and output data. ID3 grows trees splitting on all categories of an attribute, thus producing shallow and wide trees. It grows tree classifiers in three steps:

1. Splits creation in form of multiway splits, i.e. for every attribute a single split is created where attributes' categories are branches of the proposed split.
2. Evaluation of best split for tree branching based on *information gain measure*, and
3. Checking of the stop criteria, and recursively applying the steps to new branches.

These three steps are iterating and are executed in all nodes of the decision tree classifier. The information gain measure (3) is based on the well-known Shannon entropy measure (Shannon 1948) shown in (1).

$$E(S) = -\sum_{i=1}^{C} (pi \log_2 pi) \tag{1}$$

where C represents the number of classes of the output variable, and p_i the probability of the i-th class. S represents the dataset.

ID3 uses information gain (3) as a measure of split quality.

$$E\left(X, S\right) = \sum_{j=1}^{K} \left(\frac{|S_j|}{|S|} \cdot E\left(S_j\right) \right) \tag{2}$$

$$I\left(X, S\right) = E\left(S\right) - E\left(X, S\right) \tag{3}$$

where $E(X, S)$ is the expected entropy of an input attribute X that has K categories, $E\left(S_j\right)$ is the entropy of an attribute's category with respect to the output attribute, and $\left|S_j\right| / |S|$ is the probability of the j-th category in the attribute. Information gain of an attribute is the difference between entropy of the system, or node, and the entropy of an attribute. It represents the amount of information an attribute holds for the class disambiguation.

3.2 C4.5

This algorithm is an improvement of ID3. It can work with numerical input attributes as well. It follows three steps during tree growth:

1. Splits creation for categorical attributes is the same as in ID3. For numerical attributes all possible binary splits have to be considered. Numerical attributes splits are always binary.
2. Evaluation of best split for tree branching based on *gain ratio* measure, and
3. Checking of the stop criteria, and recursively applying the steps to new branches.

This algorithm introduces a new, less biased, split evaluation measure (Gain ratio). The algorithm can work with missing values, has pruning option, grouping attribute values, rules generating etc.

The Gain ratio selection criterion (4) is a measure that is less biased towards selecting attributes with more categories.

$$G(X,S) = \frac{I(X,S)}{SI(X,S)} \tag{4}$$

$$SI(X,S) = -\sum_{j=1}^{K}\left(\frac{|S_j|}{|S|}\log\frac{|S_j|}{|S|}\right) \tag{5}$$

Gain ratio divides the attribute information gain with the split info $SI(X,S)$, defined by (5), a measure that is dependent on the number of categories K in an attribute.

C4.5 can work with categorical and numerical attributes. Categorical attributes can produce multiway splits, and numerical attributes binary splits. C4.5 includes three pruning algorithms, namely reduced error pruning, pessimistic error pruning and error based pruning.

3.3 CART

The "Classification and Regression Tree" (CART) can produce classifiers and regressors. CART grows trees only with binary splits, thus producing deep and narrow trees. CART follows the following three steps:

1. Generate all possible splitting candidates. Numerical attributes splits are generated as in C4.5. For categorical attributes all possible binary groupings of attributes should be generated.
2. Evaluate the generated splits based on a chosen evaluation measure.
3. Steps 1 and 2 are repeated recursively as in all other trees until stopping criteria has been reached.

CART uses several measures for split evaluation for classification: Gini, Twoing and Ordered Twoing. We analyzed the Gini evaluation measure only. Split evaluation is based on node impurity (6). The Gini evaluation measures is shown in (7).

$$G(X,S) = i(S) - p_L * i(S_L) - p_R * i(S_R) \tag{6}$$

$$i(S) = \sum_{i,j} p(i|S)\, p(j|S), i \neq j \tag{7}$$

where $i(S)$ is calculated as the sum of products of all pairwise combination of class probabilities in a node. Probability $p(i|S)$ is the probability of class i in node S, p_L the probability of a case being sent to the left branch, p_R the probability of sending a case to the right branch, $i\ (S_L)$ the impurity of the left child node, and $i\ (S_R)$ the impurity of the right child node.

When applied as a regression tree, the mean-square error (MSE) is used to evaluate the split.

CART includes one pruning algorithm, i.e. cost complexity pruning. It can work with missing values after tree growth. CART generates surrogate splits during tree growth that can be used when cases with missing values are classified. If during classification at a tree node a case value is missing, a surrogate split is used so the classification process could go on.

3.4 CHAID

The "Chi-Squared Automatic Interaction Detector" (CHAID) decision tree algorithm with categorical data only. While creating splits, CHAID has the ability to group and ungroup attribute categories, based on chi-squared statistics, with the goal of finding the most significant splits.

CHAID has the following steps in decision tree growth:

1. Generate the most significant split for each attribute in the following way:
   1.1 Generate $2 \times C$ contingency tables (C number of classes in Y) for every pair of categories for each attribute, and perform a chi-square test for each table. For pairs of categories that do not reach a significance level threshold, merge the two least significantly different categories in a compound category and repeat this step.
   1.2 Within each compound category, which was made of three or more original categories, try to find more significant subcategories by dividing the compound category into all possible two categories divisions and find the most significant one. If the significance is beyond a defined level, implement the division and return to step 1.1.
2. Find and implement the most significant split among all attributes using chi-squared statistics and the Bonferroni multiplier for compound categories.
3. Repeat steps 1 and 2 until a user-defined stopping criteria has been reached.

Using the steps 1 and 2 CHAID can find splits that are neither binary nor multiway. It is an interesting procedure that can be used independently from CHAID in decision tree induction.

3.5 QUEST

The "Quick, Unbiased Efficient Statistical Tree" is a classification tree that works with numerical and categorical input attributes. It consists of several steps:

1. Find the most appropriate attribute $Xi(i \in 1, \ldots, m)$ for splitting with respect to Y. This is done by performing anova f-tests for numerical variables and

chi-square tests for categorical variables with respect to Y. Levene's F-test for unequal variances is computed for each ordered variable when no variable achieves a significance smaller than a predefined threshold corrected via the Bonferroni adjustment. The Bonferroni adjustment ensures that the bias is practically negligible (Loh and Shih 1997, p 826).
2. Transform all categorical variables to numerical variables with the CRIMCOORD transformation (Gnanadesikan 1977)
3. Find the split point in the selected numerical attribute $X*$. To ensure only binary splits, 2-means(Hartigan and Wong 1979) is applied to form two super classes. Afterwards, quadratic discriminant analysis is applied to find the split point.
4. Repeat steps 1 and 2 until tree is grown completely or until another user-defined stopping criteria has been reached.

QUEST includes pruning from CART, and uses a method for missing data imputation from FACT (Loh and Vanichsetakul 1988).

3.6 CRUISE

The "Classication Rule With Unbiased Interaction Detection" algorithm is an improvement of the QUEST algorithm. It consists of several steps:

1. Find the most appropriate attribute $Xi(i \in 1, \ldots, m)$ for splitting with respect to Y. The attribute selection step is done with a proposed chi-square testing framework. Five types of contingency tables for measuring chi-square statistics are used. All chi-squared statistics are afterwards normalized with the Peizer-Pratt transformation. A bootstrapping procedure is also used to eliminate the selection bias towards categorical variables. The best attribute is chosen based on the maximal normalized value from the Peizer-Pratt transformation corrected with a factor received from bootstrapping for numerical values. The five types of contingency tables are:
   a. $C \times 4$ tables for each numerical attribute where the numerical attribute is discretized in four quartiles.
   b. $C \times K$ tables for each categorical attribute.
   c. $C \times 4$ tables for each pair of numerical attributes. The four columns of the table are formed around the medians of the chosen two numerical attributes.
   d. $C \times \left(K_i^* K_j\right) (i \neq j)$ tables for each pair of categorical attributes. The four columns of the table are formed around the medians of the chosen two numerical attributes.
   e. $C \times (2K)$ for each pair of numerical and categorical attributes where the numerical attribute is discretized around their median.
2. Transform all categorical variables to numerical variables with the CRIMCOORD transformation (Gnanadesikan 1977).
3. Find the split point in the selected numerical attribute $X*$. CRUISE produces as many branches, as there are Y classes in the current node. Before applying LDA for finding the best split point, Box–Cox transformation is used for transforming $X*$ values to normal distribution, as LDA works best with normally distributed data.

4. Repeat steps 1 and 2 until tree is grown completely or until another user-defined stopping criteria has been reached.

CRUISE uses pruning and surrogate split methods like CART, and includes also missing values imputation from FACT.

3.7 GUIDE

The "Generalized, Unbiased, Interaction Detection and Estimation" tree is a regression tree. It is very similar with the CRUISE algorithm. It has the following tree growing steps:

1. Find the most appropriate attribute $Xi(i \in 1, \ldots, m)$ for splitting with respect to Y. The attribute selection step is done with a chi-square testing framework similar to CRUISE. The Y attribute is discretized into two ranges, i.e. positive valued and negative valued residuals obtained as distances of $Y's$ sample mean. Five types of contingency tables for measuring chi-square statistics are used. A bootstrapping procedure is also used to eliminate the selection bias towards categorical variables. The best attribute is chosen based on the smallest p-value for non-interacting chi-square tests. If the smallest p-value is form an interaction test, then the choice is based on minimal sum of squared errors (SSE) or the smallest p-value for interacting tests between numerical and categorical data. The five types of contingency tables used for chi-square statistics are:
   a. 2×4 tables for each numerical attribute where the numerical attribute is discretized in four quartiles.
   b. $2 \times K$ tables for each categorical variable.
   c. 2×4 tables for each pair of numerical attributes. The four columns of the table are formed around the medians of the chosen two numerical attributes.
   d. $2 \times \left(K_i^* K_j\right) (i \neq j)$ tables for each pair of categorical attributes. The four columns of the table are formed around the medians of the chosen two numerical attributes.
   e. $2 \times (2K)$ tables for all pairs of numerical and categorical attributes where the numerical attributes are discretized around the median.
2. Find the split point in the selected attribute $X*$. GUIDE has two evaluation measures for numerical splits, i.e. greedy search method that tries to finds the smallest sum of square errors (SSE) on all possible numerical splits, and median that finds the best split as the median value. Median is computationally less demanding than the greedy search method. For categorical data, binary splits are evaluated like in (Breiman et al. 1984, p 101).
3. Repeat steps 1 and 2 until tree is grown completely or until another user-defined stopping criteria has been reached.

GUIDE includes also pruning options, missing values options, and so on.

3.8 CTREE

The "Conditional Inference Tree" offers a framework for building classification and regression trees. It can work with numerical and categorical data. It can work with multivariate output data, as well. We have only analyzed the classification and regression trees with univariate output of the framework. CTREE consists of following steps:

1. The most significant attribute $Xi(i \in 1, \ldots, m)$ for splitting with respect to Y is found by first rejecting the H_0 hypothesis that there is not any relationship between input attributes and the output attribute. A permutation test linear statistic is calculated for all $X_i s$. H_0 is rejected when the minimum of the adjusted (e.g. Bonferroni multiplier) P-values is less than a pre-specified threshold. If H_0 is not rejected the tree stops to grow, otherwise the best attribute $X*$ is suggested.
2. All possible binary splits are created for $X*$.
3. The best split is chosen with a two-sample linear statistics, which is a special case of the permutation test linear statistic used in Step 1.
4. Steps 1 through 3 are repeated until tree is grown completely or until another user-defined stopping criteria has been reached (minimum number of cases in parent node, minimum sum of cases in both child nodes).

CTREE removes cases with missing values, and can create surrogate splits with the test statistic used in Step 3.

4 Components identification

There is no formal way to identify RCs. Although the literature is full of proposals what a RC is (e.g. Winn and Calder 2002), these instructions are not sufficient for RC identification. We present the sub-problems and RCs, identified in the analyzed algorithms from Sect. 3 and from several partial algorithm improvements, in Table 1. RCs belonging to a sub-problem can be used to solve the sub-problem because they have the same input-output structure. We also give sources from which we identified RCs.

In addition, we have implemented some of the identified RCs in the white-box generic decision tree framework named *WhiBo* that can be found at http://www.whibo.fon.bg.ac.rs. We use the name *WhiBo* as white-box to be distinct from the classical design approach of algorithms as black boxes.

From the analyzed algorithms, we identified the following six characteristic sub-problems in decision tree classifier induction:

1. Removing insignificant attributes,
2. Creating splits (for numerical and categorical data),
3. Evaluating a split,
4. Creating surrogate splits,
5. Stopping criteria, and
6. Pruning the tree.

These six sub-problems are discussed in the rest of this section. It is important to notice in Table 1 that some of the RCs are marked with a star (e.g. Chi-square framework for classification*). The star signifies that this RC is a complex RC consisting of several low level RCs. This issue will be addressed in the following subsections.

Table 1 Several sub-problems and RCs found in the literature

Sub-problems	Reusable component	Source	Available in WhiBo
Remove insignificant attributes	ANOVA *F* test	Loh and Shih (1997)	*X*
	Chi-square test	Loh and Shih (1997)	*X*
	Chi-square framework for classification*	Kim and Loh (2001)	
	Chi-square framework for regression*	Loh (2002)	
	Permutation test	Hothorn et al. (2006)	
Create split (numerical)	Binary	Quinlan (1993)	*X*
	Median	Loh (2002)	
	QDA*	Loh and Shih (1997)	
	LDA*	Kim and Loh (2001)	
Create split (categorical)	Binary	Breiman et al. (1984)	*X*
	Multiway	Quinlan (1993)	*X*
	Significant	Kass (1980)	*X*
Evaluate split	Information gain	Quinlan (1986)	*X*
	Gain ratio	Quinlan (1993)	*X*
	Gini index	Breiman et al. (1984)	*X*
	Twoing	Breiman et al. (1984)	
	Ordered towing	Breiman et al. (1984)	
	Distance measure	Mantaras (1991)	*X*
	Likelihood ratio chi-squared statistic	Attneave (1959)	
	Chi square test	Kass (1980)	*X*
	AUC	Ferri et al. (2002)	
	Permutation two-sample test	Hothorn et al. (2006)	
	Mean square error (MSE)	Breiman et al. (1984)	
Stop criteria	Pure node (default)	Rokach and Maimon (2008)	*X*
	Maximum tree depth	Rokach and Maimon (2008)	*X*
	Minimum cases in parent node	Rokach and Maimon (2008)	*X*
	Minimum cases in child node	Rokach and Maimon (2008)	*X*
	Minimum “evaluate split” threshold	Rokach and Maimon (2008)	
Create surrogate split	Similarity	Breiman et al. (1984)	
	Permutation two-sample test	Hothorn et al. (2006)	
Prune tree	Reduced error pruning (REP)	Quinlan (1993)	
	Pessimistic error pruning (PEP)	Quinlan (1993)	*X*
	Error-based pruning (EBP)	Quinlan (1993)	
	Cost complexity pruning (CCP)	Breiman et al. (1984)	

4.1 Removing insignificant attributes

There are numerous ways features (attributes) can be selected. An extensive list of methods can be found in (e.g. Rokach and Maimon 2008). Instead of using feature selection before the decision tree algorithm is applied on data, it is also possible to use feature selection at each node during tree growth. Decreasing the number of candidate attributes improves computational efficiency of an algorithm. The idea for this sub-problem was found in QUEST (Loh and Shih 1997) where chi-square test and ANOVA F test were used for categorical and numerical attributes respectively to find the most significant attribute which will be split further. Several other algorithms include this sub-problem to find the most significant attribute (e.g. CRUISE, GUIDE, and CTREE).

We propose the extension of the usage of RCs “ANOVA F-test” and “Chi-square test” to opt out attributes that do not satisfy a predefined significance threshold. This RC is implemented in *WhiBo* and allows the original usage (like in QUEST) and the extended usage (remove attributes that do not satisfy a threshold, or remove the defined percentage of attributes). In Delibasic et al. (2010) experimental evidence is presented how the usage of these RCs, while improving the computational speed, has effect on classification accuracy.

CRUISE and GUIDE propose a chi-square test framework where five types of contingency tables should be evaluated with the chi-square statistic. Also, a bootstrapping procedure is suggested to find a parameter by which statistics for numerical data should be multiplied to reduce the bias towards categorical data selection. CRUISE and GUIDE use similar contingency table design, and differences are due to CRUISE being a classification algorithm, and GUIDE a regression algorithm. Both CRUISE and GUIDE propose selection of the most significant attribute that include several procedures on a lower level of granulation. These procedures, if seen as RCs, could be easily exchanged with other RCs solving the same sub-problem on a lower level of decomposition. Five RCs can easily be identified within the complex (starred) RC “Chi-square framework for classification”. These are: “Create bootstrap samples”, “Design contingency tables”, “Chi-square test”, “Peizer-Pratt transformation”, and “Calculate bias correction”. For gaining full advantages of component-based design, a generic structure should be defined that could allow exchange of RCs on a lower level of decomposition. Currently *Whibo* does not support this. The complex RC “Chi-square framework for classification” can, however, be used as a black-box within the proposed framework.

Component-based design could, in the aforementioned RC, allow better analysis of the RCs on the lower level. It could give an answer whether the design of contingency tables is optimal, and whether there is a better way to propose bias correction than with the bootstrapping procedure proposed inside this RC. We will show an example of the decomposition on a lower lever of granulation in Sect. 4.2.

An important idea of component-based design is to identify atomic RCs that have a much wider area of application, than complex RCs. Following is an example of how the atomic “ANOVA F-test” RC can be described with Tracz (1990) definition.

Component Name: "ANOVA F test"

1. Concept:
   Description: Measures if there are significant differences in values of a numerical attribute on categories of a categorical attribute.
   Input: Dataset with one or more numerical attributes and one categorical output attribute, significance threshold.
   Output: Significance levels for numerical attributes, Selected numerical attributes.
2. Context:
   Application: Tests if values of the numerical attributes are significantly different when compared to other attribute categories. It does not test whether a significant attribute includes a significant split. It is merely a heuristic that can help to reduce the number of attributes that will be used for finding the best split.
3. Content:
   Uses the ANOVA F-test to measure the significance of an attribute like in (Wu and Flach 2002).

4.2 Creating a split

The problem of choosing how to grow a tree is important (Kim and Loh 2001;Utgoff et al. 1997). There is, in general, no best way to split a tree (Kim and Loh 2001). The same study reports that on some datasets multiway splits can be beneficial, where they can provide more interpretable results than binary splits. Binary splits are computationally more demanding than multiway splits, produce trees with more depth but can produce, on some datasets, more accurate classifiers. It is not easy to determine the right way of splitting attributes.

We analyzed splits for categorical and numerical attributes. In ID3 and C4.5 this process is done by making all attribute categories branches of a split, i.e there is one possible split for each categorical attribute. If a categorical attribute consists of K categories, it can be split binary in $(2^{K-1} - 1)$ ways, multiway in one way, but there is also the possibility to group similar categories producing neither binary nor multiway splits, like in CHAID, or, on the other hand, use several ways to produce splits (e.g. generate multiway and binary splits at the same time as candidate splits).

In the decision tree algorithms we've analyzed, we identified several RCs for splitting categorical attributes. We found "Binary" splits (CART, GUIDE, and CTREE), "Multiway" splits (ID3, C4.5) or "Significant" splits (CHAID).

Numerical attributes are usually splitted binary, although there are differences how binary splits are produced. Algorithms like C4.5 and CART generate binary splits only on points where optimal splits can exist, as proved in Fayyad and Irani (1992). On the other hand, algorithms like QUEST, and CRUISE generate splits directly. QUEST uses quadratic discriminant analysis (QDA) to generate the best split. Because QDA produces as many splits as there are classes in Y, the Y attribute is beforehand transformed into two super-classes with 2-means (Hartigan and Wong 1979). CRUISE generates splits with linear discriminant analysis, thus producing as many splits as there are classes in Y at the current node. Although both QUEST and CRUISE propose procedures for splits creation, we see them as non-atomic, complex RCs that can theoretically fit the proposed generic decision tree framework. Still, these procedures consist of several atomic procedures that could, on a lower level of granulation, easily be exchanged with other RCs having the same functionality. E.g. The complex RC "QDA*" consists of three RCs: "CRIMCOORD transformation" (transforms all numerical attributes to categorical attributes), "2-means" (groups Y classes into 2 super-classes), and "QDA" (is used to find the best split). Three sub-problems can be identified in this procedure: "Transform categorical to numerical attribute", "Group classes", and "Create discriminant splits". The RCs currently used for solving these sub-problems could be easily exchanged with other RCs providing the same functionality within a component-based framework.

Following is an example of how the "Significant" RC from CHAID can be described with Tracz (1990) definition.

Component Name: "Significant"

1. Concept:
   Description: Groups similar categories into mergers that can produce significant splits.
   Input: Categorical input attributes and categorical output attribute.
   Output: Merged categories within input attributes.
2. Context:
   Application: Can be used when it is important to join similar categories into a merger category. For categorical attributes with large number of categories this way of grouping categories can produce more interpretable results.
3. Content:
   Uses a method for grouping categorical attributes categories into merged categories as described in Kass (1980).

4.3 Evaluating a split

Every split has to be evaluated using some evaluation measures. Various split evaluation measures can be found in literature. ID3 uses "Information gain", C4.5 "Gain ratio", CART uses "Gini", and "Twoing" for classification, and "Mean square error" (MSE) for regression. CHAID uses "Chi-square test". GUIDE uses also MSE for split evaluation for regression trees. CTREE uses "Permutation two-sample test" for both regression and classification split evaluation. After a split is created it can be evaluated with one of these measures.

Some properties of splitting measures were discussed by Breiman (1996), Mingers (1989), Fayyad and Irani (1992), Loh and Shih (1997), Shih (1999), Lim et al. (2000) etc. E.g. "Gini" does not try to balance the size of the child nodes, while "Twoing" and entropy measures do (Breiman 1996). "Information gain" is biased towards the number of categories in an input attribute. "Gain ratio", on the other hand, is a less biased evaluation measure that is based on information gain (Quinlan 1993).

We analyzed also partial algorithm improvements. In Mantaras (1991) the "Distance measure" is proposed. It is unbiased towards the number of categories in an attribute and shows, in general, better results than "Information gain". The "Likelihood ratio chi-squared statistic" is found in Attneave (1959) where it is used to measure statistical significance of the “Information gain” measure. The AUC (Area under curve) measure is found in Ferri et al. (2002) and outperforms “Gain ratio” and “Gini” on some datasets.

It is difficult to find the best evaluation split, however one can find evaluation measures that behave better on certain datasets than others.

4.4 Stopping criteria

A decision tree grows by implementing new splits in the existing tree. Trees grow, in general, until all nodes become pure. However, there are also other possibilities to stop tree growth. In fact, there are many ways a tree can be forced to stop growing. The methods for preventing further tree growth are also called pre-pruning methods.

Every decision tree algorithm we analyzed had stopping criteria. The stopping criteria should guarantee that a split is, in some way, significant.

In most decision trees the following stopping criteria can be found (Rokach and Maimon 2008):

1. All cases in a node have the same class value of the output attribute. (“Pure node” from Table 1),
2. A maximum predefined tree depth has been reached (“Maximum tree depth”),
3. A minimum of cases in the parent node has been reached (“Minimum cases in parent node”),
4. A minimum of cases in the child node has been reached (“Minimum cases in child node”), and
5. A minimum evaluation measure threshold has been reached (Minimum “evaluate split” threshold).

Breiman et al. (1984) and Quinlan (1993) suggest that is not appropriate to use stopping, i.e. prepruning, criteria during tree growth because this influences the quality of a classifier. Instead, they recommend first growing the tree, and afterwards pruning the tree. On the other hand Hothorn et al. (2006), suggest that using significance tests as stopping criteria (i.e. prepruning) could generate trees that are similar to optimally pruned trees.

4.5 Creating a surrogate split

After the classifier is built, it can happen that it should be applied on cases with missing values. If the original node can’t be used for classification, a surrogate node has to be used. We identified in CART (Breiman et al. 1984) a way of creating surrogate splits during tree growth that we named “Similarity”. This RC finds one or more splits that should be used when the original value is missing in a case during classification. For every node several surrogate splits can be created, and they are ordered with respect to how they are similar with the original split.

In CTREE we identified a RC for creating surrogate splits “Permutation two-sample test”.

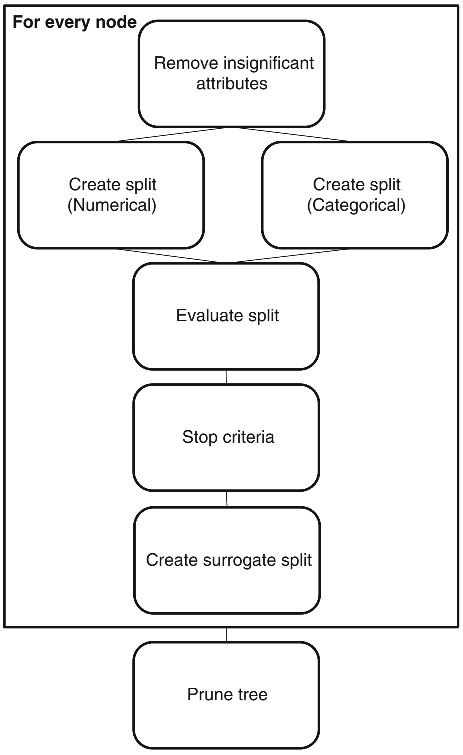

Fig. 1 The generic decision tree

4.6 Pruning the tree

Grown trees are often pruned, so their complexity is reduced while maintaining accuracy within certain borders. "Prune tree" algorithms can find a sub-tree of the grown tree that achieves the best compromise between the quality of the tree and the complexity of the tree. Less complex trees are desired for two reasons, their greater interpretability, and their better generalizations.

There are a lot of pruning techniques: reduced error pruning, pessimistic error pruning and error-based pruning (Quinlan 1987), cost-complexity pruning (Breiman et al. 1984) etc. Generally, pruning techniques can improve accuracy, but the effects depend on domain (Malerba et al. 1996). All pruning algorithms can be used on all decision trees, so they are independent of the algorithm they originate from.

5 Generic decision tree

The components we've described can be shown in a form of a generic decision tree. The generic tree we propose is shown in Fig. 1. The generic tree is shown in a form that looks like an algorithm; however it isn't. The elements of the generic decision tree are structural in a sense that they define the building blocks of an algorithm. They

Table 2 The generic decision tree algorithm

```
Input: Dataset
Output: Decision tree model
Step 1. Optionally, use "Remove insignificant attributes"
to eliminate uninformative attributes in current
tree node.
Step 2. Use "Create split" to create candidate tree split in
current tree node.
Step 3. Use "Evaluate split" to measure "goodness" of
candidate split.
Step 4. If candidate split is better than best split,
remember new candidate split as best.
Step 5. Repeat steps 2–4 until no more candidate splits
are produced by "Create split".
Step 6. Optionally, use "Create surrogate split" to create
n surrogate splits for the current node.
Step 7. If "Stop criteria" is met, create leaf node and add
it to the tree model.
Otherwise split the dataset according to the best
split, and recursively return to step 1 for each
new branch of the node.
Step 8. Optionally, after the tree model is built use "Prune
tree" to shorten branches which are
uninformative according to prune criteria.
```

are identified as important sub-problems in decision tree induction. The RCs, on the other hand, define the solution for a sub-problem.

The generic tree proposed in Fig. 1 has crucial and optional sub-problems for a type of algorithm. To build a C4.5-like classifier it is necessary to define RCs for "Create split", "Evaluate split", and "Stop criteria" ("Pure node" is always used). It is optional to use "Remove insignificant attributes", "Create surrogate split", and "Prune tree".

To build an algorithm from the selected RCs it is necessary to define a generic decision tree algorithm that couples RCs together. We propose a generic decision tree algorithm in Table 2.

We will show further how original algorithms can be reproduced, how they can be modified or extended, and how new decision tree algorithms for classification can be built with the proposed generic tree.

In Fig. 2 the C4.5 decision tree algorithm is composed using following RCs for sub-problems:

"Remove insignificant attributes" = **none**
"Create split (Numerical)" = **"Binary"**
"Create split (Categorical)" = "**Multiway"**

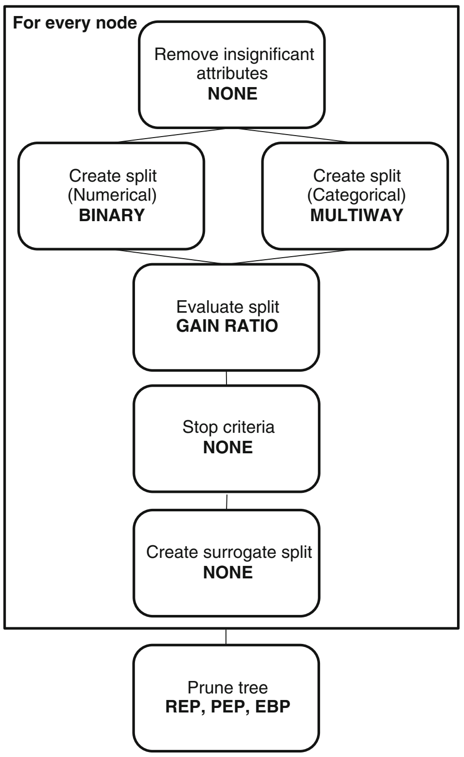

Fig. 2 C4.5 in the generic decision tree

"Evaluate split" = "**Gain ratio"**
"Stop criteria" = **none** ("Pure node" is used by default)
"Create surrogate split" = **none**
"Prune tree" = "**REP**", "**PEP**" or "**EBP"** (only one can be used at a time)

All algorithms can be extended in some way. For example, ID3 can be easily extended by including "Create surrogate split" (Fig. 3). New algorithms can be created by combining the analyzed components. Including "Remove insignificant attributes" could improve all the original algorithms.

We can easily imagine an algorithm that uses "Remove insignificant attributes" RCs "ANOVA *F* test" and "Chi-square test", uses for "Create split (numerical)" "Binary", for "Create split (Categorical)" "Significant**"**, for "Evaluate split" "Gain ratio", and uses "Stop criteria" "Maximum tree depth" and "Minimum cases in child node", "Error-based pruning" for "Pruning", and "Significant" for "Create surrogate split" (Fig. 4).

These new algorithms are not intended to replace the well-known algorithms, because they can not outperform them on every dataset. Component-based design, as stated in Delibasic et al. (2010), suggests that "for a specific dataset we should search for the optimal component interplay instead of looking for the optimal among predefined algorithms".

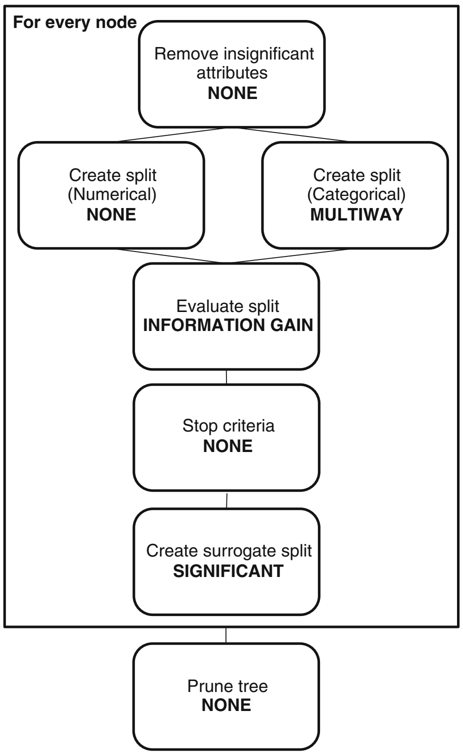

Fig. 3 ID3 with "Create surrogate split" in the generic decision tree

All of the analyzed algorithms can be implemented within this framework. This way of algorithms design allows fast creation of new algorithms, that otherwise would need more time to implement. Some component-based algorithms and their performances are presented in Delibasic et al. (2010). There are a lot of RCs that are tied to their algorithms, but could otherwise be used smartly in decision trees. One of them is certainly the "Significant" RC from CHAID that is an interesting, yet somehow forgotten way to generate splits.

6 Conclusion and further research

In this paper we outlined our ongoing study aimed at improving design of decision tree algorithms. Although only a few decision tree algorithms were discussed, the approach is applicable to other decision tree based algorithms and the idea of component-based design can be also extended to other machine learning algorithms (e.g. partitioning clustering (Delibasic et al. 2009).

The main contribution of the proposed investigation is in:

1. An improved characterization of decision tree algorithms that allows easier analysis of their advantages and disadvantages. This kind of lower resolution analysis,

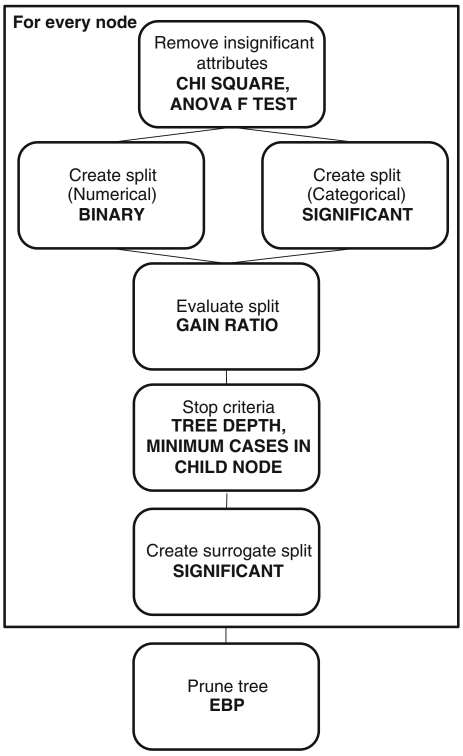

Fig. 4 A new algorithm in the generic decision tree

we believe, is especially useful when aimed at developing new decision tree based classifiers tailored for certain application types.

2. Easier development of new decision tree based algorithms that are constructed using RCs found in existing and well known algorithms. It can happen that none of existing algorithms is fully appropriate for solving a particular problem. However, perhaps appropriate components can be found in different algorithms. The proposed framework provides a fairly general and simple environment for components exchange to better adapt to new demands.

To allow a community-wide evaluation of the proposed approach for designing data mining algorithms based on reusable components we have implemented the proposed decision tree and some of the RCs presented in this paper as a plug-in in the open-source machine learning platform *RapidMiner*. Our proposed generic tree, named *WhiBo*, and instruction of how to install, use, and extend it are available at http://www.whibo.fon.bg.ac.rs.

The idea is to enable testing of individual contributions (RCs) in a more controlled environment, as well to enable creating tightly tailored algorithms for a specific need.

In further research we plan to use *WhiBo* software for more thorough analysis of the identified components as to improve current understanding of RCs behavior under various conditions. We hope that our further and a community-wide evaluation can lead towards more automatized selection of RCs based on inherent properties of data (e.g. type, distribution, attribute correlation, quality of data etc.) and user constraints.

Acknowledgments This research was supported by a grant from the Serbian Ministry of Science, TR 12013.

References

Adams M, Coplien J, Gamoke R et al (1991) Fault-tolerant telecommunication system patterns. In: Rising L (ed) The pattern handbook: techniques, strategies, and applications. Cambridge University Press, New York, pp 189–202

Alexander C (1979) The timeless way of building. Oxford University Press, New York

Attneave F (1959) Application of information theory to psychology. Holt, Rinehart and Winston, New York

Breiman L, Friedman J, Stone CJ et al (1984) Classification and regression trees. CRC Press, Boca Raton

Breiman L (1996) Technical note some properties of splitting criteria. Mach Learn 24:41–47

Coplien JO, Schmidt DC (1995) Pattern languages of program design. Addison-Wesley Professional, Reading

Coplien JO, Harrison NB (2005) Organizational patterns of agile software development. Prentice Hall PTR, Upper Saddle River

Delibasic B, Kirchner K, Ruhland J (2008) A pattern-based data mining approach. In: Preisach C, Burckhardt H, Schmidt-Thieme L et al (eds) Data analysis, machine learning and applications. Springer, Berlin, pp 327–334

Delibasic B, Kirchner K, Ruhland J, Jovanovic M, Vukicevic M (2009) Reusable components for partitioning clustering algorithms. Art Int Rev 32:59–75

Delibasic B, Jovanovic M, Vukicevic M, Suknovic M, Obradovic Z (2010) Component-based decision trees for classification. Intell Data Anal 15(5): accepted for publication

Drossos N, Papagelis A, Kalles D (2000) Decision tree toolkit: a component-based library of decision tree algorithms. In: Zighed DZ, Komorowski J, Zytkow J (eds) Principles of data mining and knowledge discovery. Springer, Berlin, pp 121–150

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

Ferri C, Flach P, Hernandez-Orallo J (2002) Learning decision trees using the area under the ROC curve. In: Sammut C, Hoffmann A (eds), Proceedings of the 19th international conference on machine learning, Morgan Kaufmann, pp 139–146

Freeman P (1983) Reusable software engineering: Concepts and research directions. In: Workshop on reusability in programming, ITT Programming, Stratford, Conn., pp 2–16

Gamma E, Helm R, Johnson R et al (1995) Design patterns: elements of reusable object-oriented software. Addison, Reading

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

Hartigan JA, Wong MA (1979) Algorithm 136. A k-means clustering algorithm. Appl Stat 28:100

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

Kass GV (1980) An exploratory technique for investigating large quantities of categorical data. Appl Stat 29:119–127

Kim H, Loh WY (2001) Classification trees with unbiased multiway splits. J Am Stat Assoc 96:598–604

Lea D (1994) acs.pdf. In: Design patterns for avionics control systems. http://gee.cs.oswego.edu/dl/acs/acs.pdf

Lim TS, Loh WY, Shih YS (2000) A comparison of prediction accuracy, complexity, and training time of thirty-three old and new algorithms. Mach Learn 40:203–228

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

Loh WY (2002) Regression trees with unbiased variable selection and interaction detection. Stat Sin 12:361–386

Malerba D, Esposito F, Semeraro G (1996) A further comparison of simplification methods for decision tree induction. In: Fisher D, Lenz HJ (eds) Learning from data: AI and statistics V. Springer, Berlin, pp 365–374

Mantaras RL (1991) A distance-based attribute selection measure for decision tree induction. Mach Learn 6:81–92

Mierswa I, Wurst M, Klinkenberg R, Scholz M, Euler T (2006) YALE: Rapid Prototyping for Complex Data Mining Tasks. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, New York, pp 935–940

Mingers J (1989) An empirical comparison of pruning methods for decision tree induction. Mach Learn 4:227–243

Murthy SK (1998) Automatic construction of decision trees from data: A multi-disciplinary survey. Data Min Knowl Discov. doi:10.1023/A:1009744630224

Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106

Quinlan JR (1987) Simplifying decision trees. Int J Man Mach Stud 27:221–234

Quinlan JR (1993) C4.5 Programs for machine learning. Morgan Kaufmann

R Development Core Team (2008) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna

Rokach L, Maimon O (2008) Data mining with decision trees—theory and application. World Scientific Publishing, London

Safavian SR, Landgrebe D (1991) A survey of decision tree classifier methodology. IEEE Trans Syst Man Cybern 21:660–674

Shannon CE (1948) shannon1948.pdf. A mathematical theory of communication. http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

Shih L (1999) Families of splitting criteria for decision trees. Stat Comput 9:309–315

Siddique NH, Amavasai BP, Ikuta A (eds) (2007) Special issue on hybrid techniques in AI. Artificial Intelligence Revue 27

Sommerville I (2004) Software engineering. Pearson, Boston

Sonnenburg S, Braun ML, Ong CS, Bengio S, Bottou L, Holmes G, Le Cun Y, Mueller KR, Pereira F, Rasmussen CE, Raetsch G, Schoelkopf B, Smola A (2007) The need for open source software in machine learning. J Mach Learn Res 8:2443–2466

Tracz W (1990) Where does reuse start? ACM SIGSOFT Softw Eng Notes 15:42–46

Utgoff PE et al (1997) Decision tree induction based on efficient tree restructuring. Mach Learn 29:5–44

Winn T, Calder P (2002) Is this a pattern? IEEE Softw 19:59–66

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. 2. Morgan Kaufmann, San Francisco

Wu S, Flach PA (2002) Feature selection with labeled and unlabeled data. In: Proceedings of ECML/PKDD workshop on integration and collaboration aspects of data mining, decision support and meta-learning, IDDM 2002, Helsinki

Zeileis A, Hothorn T, Hornik K (2008) Model-based recursive partitioning. J Comput Gr Stat 17(2):492–514