Università degli Studi di Napoli
Federico II

Tree Based Methods for Data Editing and Preference Rankings

Antonio D'Ambrosio

Tesi di Dottorato di Ricerca in
Statistica

XX Ciclo

Dipartimento
di Matematica e Statistica
Università degli Studi di Napoli "Federico II"
via Cintia, Monte Sant'Angelo – 80126 Napoli

Tree Based Methods

for Data Editing and Preference Rankings

Napoli, 30 novembre 2007

Ringraziamenti

É difficile in poche righe ricordare e ringraziare tutte le persone che hanno contribuito a rendere "migliori" questi ultimi anni. Mi reputo davvero fortunato se posso dire che, davvero, sono tante.
I miei piú sentiti ringraziamenti vanno al prof. Roberta Siciliano, mia guida in questo mio percorso di ricerca. Se non mi sono mai sentito solo, è soprattutto grazie a lei.
Un sincero ringraziamento va al prof. Natale Carlo Lauro, che con i suoi sempre puntuali ed opportuni consigli ha saputo fornirmi spunti di riflessione rivelatisi importantissimi.
Grazie al dott. Massimo Aria, che ha contribuito in maniera determinante ad introdurmi nel mondo della statistica computazionale; se ho cominciato a "sbirciare" in ambiente MatLab é merito suo.
Grazie al prof. Antonio Mango, che mi ha insegnato che la ricerca é confronto continuo.
Grazie a tutti i membri del Dipartimento di Matematica e Statistica, ai colleghi (e soprattutto amici) dottorandi con i quali ho passato momenti indimenticabili.
Un ringraziamento particolare va al prof. Jaromír Antoch che mi ha insegnato (ma é dura da imparare) cosa vuol dire avere metodo, al prof. Ab Mooijaart che mi ha insegnato molto in ambiente MatLab, al prof. Willem Heiser che mi ha introdotto nel mondo dei "preference rankings".

Ma sopratutto un ringraziamento va a Brunella, mia moglie, la persona che mi ha sopportato e supportato con infinita pazienza ed infinito amore e che non saró mai in grado di ripagare, e a Debora e Luca, i miei fratelli, che mi sono stati sempre accanto. Dove starei senza di te, dove starei senza di voi...

ai miei genitori

Contents

List of Tables

List of Figures

Introduction

This work is focused on tree-based methods as:

- complete data pre-processing methods (Data Editing), by handling missing data imputation problems, data validation and completion of data coming from different sources (Data Fusion);
- new way in the framework of supervised classification in terms of preference rankings.

Data Editing can be understood as the methods used to edit (e.g., clean-up) and impute (fill-in) missing or contradictory data. In this area segmentation methodologies have a relevant role because they can be used in every foreseen step: data validation, missing data imputation and data fusion, which is a very special case of missing data imputation. The strength of these techniques, as well known, is that they are non-parametric, i.e. they work regardless to the nature of the variables composing the dataset (qualitative or quantitative), they can be used as learning algorithm forming ensemble classification rules (Boosting, Bagging, Random Forest).
In the framework of preference rankings, the evaluation of the interactions of a set of predictors associated to a ranking matrix is a topic dealt only with log-linear models. Treatment of preference rankings response variables with classification trees is a novelty in the literature.

This thesis is structured in five chapters.

The knowledge discovery process from databases fits the scientific paradigm of Statistical Learning: Data Mining, Inference and Prediction (Hastie, Tibshirani, Friedman, 2001), representing also the starting point for the combination of modern statistics and computer science for Intelligent Data Analysis (Hand, Berthold, 2005).
First chapter is about tree-based models and their methodological context. This chapter deals also with ensemble methods, which are a classifiers aggregation procedure with the aim of building an aggregated more accurate classifier than every single classifiers with which it is made. In particular, Bagging and AdaBoost algorithms and their variants for both regression and classification problems are described.

Second chapter is about Data editing. Knowledge Mining refers to the statistical and automatic learning finalized to the knowledge discovery from databases through Data Mining and Predictive Learning in combination with computer science such to exploit the use of available and upgraded information. Data Editing is a preliminary step of Knowledge Mining, such to obtain a database characterized by homogeneous, complete, coherent, and, in general, validated data from the quality point of view.
Indeed Statistical Data Editing (SDE) is defined as the set of methods used to edit and impute missing or contradictory data. The final result of SDE is obtaining data that can be used for analytic purposes. These include primary purposes such as estimation of totals and subtotals for publications that are free of self-contradictory information.

Third chapter is about missing data imputation. The starting point is the incremental missing data imputation philosophy which is based on a definition of a lexicographic order that indicates the row and column position of a missing value in the data matrix. The imputation is incremental because, step by step, data are updated using information which were missing in the previous steps to impute data in the next stage. In this chapter, a lexicographic order only of the columns is defined, and the imputation is made "by column" instead of for single missing-ness. This column, in which there are missing values, at turn plays a role of response variable for an ensemble procedure which uses a STUMP for binary imputation variable case, or both a FAST regression and classification tree for the imputation of numerical or multiclass missing data respectively. Strong point of this chapter is the BINPI (Boosted Incremental Non Parametric Imputation) algorithm: this is tested on several simulated and real data sets from which data were deleted according a Missing At Random schema and then used to evaluate the goodness of imputation.

Within the fourth chapter an algorithm for Data Fusion imputation is proposed. Data Fusion is considered a very special case of missing data imputation, with the difference that data coming from two different and independent sources. A data matrix block is common to both sources, whereas one block is specific for only one of them. Particularity in Data Fusion is that data are missing because never they have been collected. Algorithm R.T.I.I. (Robust Tree-based Incremental Imputation) is the proposed solution for data imputation for Data Fusion. Results of the proposed algorithm, in comparison with other classic techniques used for the same problem, show as R.T.I.I. algorithm contributes to re-build missing variables in terms of both mean and variance.

In the fifth chapter a Tree-Based Multivariate Tree for Rankings is defined.
In the framework of preference rankings, interactions among predictors which are able to explain a supervised classification when the latter is characterized by preference rankings as response variable is a nov-

elty. Such a problem is dealt in literature only with log-linear models. In this context, the definition of a tree-based structure can extend to preference rankings response variable the ease of interpretation typical of tree-based models. Conceptually, a rank can be considered as a unique multidimensional "entity", so that the techniques known in literature to define split for multivariate response variables are not able to obtain impurity measure which are feasible in this case. The idea is to define an impurity measure based on a suitable distance for rankings, taking in account that in the more realistic cases, ties are allowed. Metric chosen is the Kemeny distance, and the impurity is defined as the sum of Kemeny distances within node.

Chapter 1

Tree-based methods and methodological context

1.1 Classification and Regression Trees

Binary segmentation procedure consists of a recursive binary partition of a set of objects described by some explanatory variables (either numerical or and categorical) and a response variable. In the following, CART procedure [16] is followed.
The data are partitioned by choosing at each step a variable and a cut point along it according to a goodness of split measure which allows to select that variable and cut point that generates the most homogeneous subgroups respect to the response variable. The procedure results in a nice and powerful graphical representation known as decision tree which express the sequential grouping process. Because of the evident analogy with the graph theory, a subset of observations is called node and nodes that are not split are called terminal nodes or leaves (see figure 1.1). Each node has a number such that generic node t generates the left node $2t$ and the right node $(2t+1)$. This approach

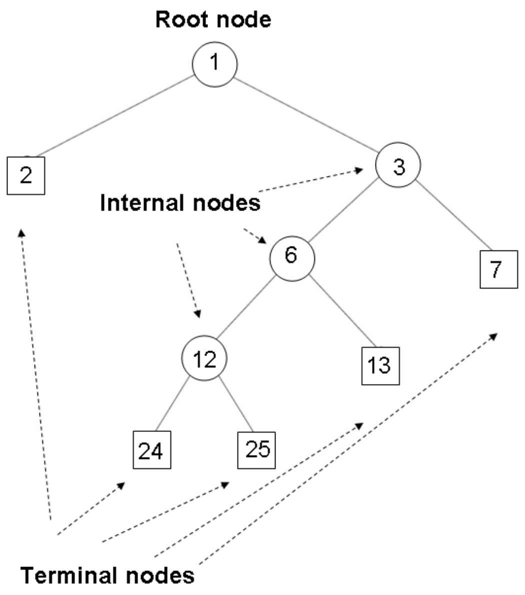


Figure 1.1: Tree-based structure

was proposed by authors of statistical software SPAD (Cisia Institute, France). In this way, it is always possible to recognize the position of each node given its number deriving the path from the node to the root node and vice versa. In example, in the above figure, the node 6 is the left node of its parent node 3 which is the right node of its parent node 1 (the root node).
Once the tree is built, a response value or a class label is assigned to each terminal node. According to the nature, categorical or numerical, of the response variable, in the framework of binary segmentation procedures a distinction is made between Classification Tree (for the

categorical response case) and Regression Tree (for the numerical response case). In classification tree case, when the response variable takes value in a set of previously defined classes, the node is assigned to the class which presents the highest proportion of observations (by voting); whereas in the regression tree case, the value assigned to cases in a given terminal node is the mean of the response variable values associated with the cases belonging to the given node. In both cases this assignment is probabilistic, in the sense that a measure of error is associated to it.
The main aim of the procedure is to define a classification/prediction rule on the basis of a learning set (also called training set), for which the values of a response variable Y, and of a set of K explanatory variables $(X_1, \ldots, X_k, \ldots, X_K)$ (either numerical or/and categorical) have been recorded.
The recursive partitioning procedure follows a *divide and conquer* algorithm, in the sense that in principle the algorithm continues partitioning nodes until all leaves contain a single case or cases either belonging to the same class or presenting the same response value. This leads to overlarge trees with many rules which are hard to understand and overfit the data.
In practice, when performing binary segmentation one has to look for a compromise that allow for the trade-off between the *exploratory* and the *confirmatory* purposes of the tree structures methodology. A distinction is made between the two problems involved in investigating the data sets: that is, whether to explore dependency, or to predict and decide about future responses on the basis of the selected predictors.
Explanation can be obtained by performing a segmentation of the objects until a given stopping rule defines the final partition of the objects to interpret.
Confirmation is a completely different problem that requires the definition of decision rules, usually obtained by performing a *pruning*

procedure soon after a segmentation procedure. Therefore, a further step, tree simplification, is usually carried out to avoid *overfitting* and improve the understandability of the tree by retrospectively pruning some of the branches.
Summarising, tree based methods involve the following steps:

- the definition of a splitting criterion;
- the definition of a stopping rule;
- the definition of the response classes/values to the terminal nodes;
- tree pruning, aimed at simplifying the tree structure, and tree selection, aimed at selecting the final decision tree for decisional purposes

1.1.1 Splitting criteria

Let (Y, X) be a multivariate random variable where X is a set of K categorical or numerical predictors $(X_1, \dots, X_k, \dots, X_K)$ and Y is the response variable. The first problem in tree building is how to determine the binary splits of the data into smaller and smaller subgroups. Since the partitioning is just two branches, splitting variables need to be created from the original explanatory variables. Accordingly, data partitioning is based on a set of Q binary questions of the form:

$$\text{is } X_k \in A?,$$

so that, if X_k is categorical, A includes subsets of levels, while if X_j is numeric, Q includes all questions of the form:

$$\text{is } X_k \leq c?,$$

for all c ranging over the domain of X_k. For example, if $K = 3$, X_1, X_2 are numerical and $X_3 \in \{a_1, a_2, a_3\}$, Q includes all questions of the form:

$$X_1 \leq 3.5?$$
$$X_2 \leq 5?$$
$$X_3 \in \{a_1, a_3\}\,?$$

The set of possible splitting variables is finite and the number of splitting variables that can be created from a given explanatory variable depends on the type of variable, i.e., according to its measurement. Table 1.1 reports the number of splitting variables that can be generated by any type of explanatory variable according to its scale of measurement. To each tree node the algorithm generates all the possible splitting variables and searches through them one by one, so the easiest case to deal with is binary variables that can generate just a single splitting variable, numeric variables are treated as ordered with N categories. Finally, unordered variables are the most difficult to deal with because they can generate a very large number of splitting variables even for a small value of M. Once that, at a given node, the set of binary questions has been created, some criterion which guides the search in order to choose the best one to split the node is needed. As said before, the key idea is to split each node so that each descendant is more homogeneous than the data in the parent node. To reach this aim, we need a measure of homogeneity to be evaluated by means of a *splitting criterion*. In the CART methodology the idea of finding splits of nodes which generate more homogeneous descendant

Explanatory variable	**Categories**	**# of splitting variables**
Numeric	N	$N-1$
Binary	2	1
Ordered	M	$M-1$
Unordered	M	$2^{M-1}-1$

Table 1.1: Origin of the splitting variables

nodes has been implemented for classification trees by introducing the so called *impurity function*.
Let $p(j|t) \geq 0$ be the proportions of cases in node t belonging to class j with $\sum_{j=1}^{J} p(j|t) = 1$.
An impurity function ϕ is a function of the set of all J-tuples of numbers $p(j|t)$ with the properties ([16], pag 24):

1. ϕ is maximum only at the point $\{1/J, 1/J, \ldots, 1/J\}$;

2. ϕ achieves its minimum only at the points $(1, 0, \ldots, 0), (0, 1, \ldots, 0), (0, 0, \ldots, 1)$;

3. ϕ is a symmetric function of $p(j|t)$.

There are several impurity functions satisfying these three properties. The most common are:

1. the error rate, or the misclassification ratio:

$$i(t) = 1 - max_j p(j|t)$$

2. the Gini diversity index

$$i(t) = 1 - sum_j p(j|t)^2$$

3. the entropy measure

$$i(t) = -sum_j (pj|t) log(j|t)$$

Talking about regression trees, the splitting criterion is based on the search of that split that generates the most different descendant nodes in terms of mean value of the response variable.

$$i\,(t) = \frac{1}{N} \sum\nolimits_{x_n \in t} (y_n - \bar{y}_t)^2 \tag{1.1}$$

which can be meant as the total sum of squares (TSS), divided by N, where N is the sample size, $\bar{y}_t = \frac{1}{N_t} \sum\limits_{x_n \in t} y_n$, N_t is the total number of cases in the node t where the sum is over all y_n such that $x_n \in t$. If s is a proposed split of a generic node t into two offspring t_l and t_r , and p_l and p_r are the proportions of objects in node t which the split s puts into nodes t_l and t_r respectively, then a measure of the change in impurity which would be produced by split s of node t is given by:

$$\Delta i(t,s) = i(t) - [i(t_l)p_{t_l} + i(t_r)p_{t_r}] \tag{1.2}$$

Δi, called decrease in impurity, can be used as splitting criterion: a high value means that a proposed split is a good one. At a given node t, a split s^* maximising equation 1.2 is optimal and used for generate two descendants t_l and t_r . Let $\tilde{T}$ be the set of all terminal nodes of the tree T: the total impurity of any tree T is defined as

$$I\left(T\right) = \sum_{t \in \tilde{T}} i\left(t\right) p(t)$$

To proceed with tree growing, CART procedure must compute the decrease in impurity associated to each possible split generated by each variable. For example, suppose to have a binary response variable and a set of six predictors as defined in table 1.2. In the root node the number of decreases in impurity to be computed is 31+63+2+1+4+3 = 104. Therefore, computational cost of CART is really high, because this procedure must be repeated until a stooping rule in tree-building occurs.

1.1.2 Two Stage splitting criterion

Mola and Siciliano [86, 85] have proposed a Two-Stage splitting criterion to choose the best split. This approach relies on the assumption

Variable	Nature	Categories	# of split
X_1	*Nominal*	6	$2^5 - 1 = 31$
X_2	*Nominal*	7	$2^6 - 1 = 63$
X_3	*Ordinal*	3	$3 - 1 = 2$
X_4	*Binary*	2	1
X_5	*Ordinal*	5	$5 - 1 = 4$
X_6	*Ordinal*	4	$4 - 1 = 3$

Table 1.2: Example of generation of splits according to the nature of the predictors

that a predictor X_k is not merely used as a generator of partitions but it plays also a global role in the analysis. In the first stage, a variable selection criterion is applied to find one or more predictors that are the most predictive for the response variable. On the basis of the set of partitions generated by the selected predictor(s), a partitioning criterion is considered in the second stage in order to find the best partition of the objects at a given node. The criteria to be used in the two stages depends on the nature of the variables, the tool of interpretation and the desired description in the final output. The partitioning algorithm takes account of the computational cost induced by the recursive nature of the procedure and the number of possible partitions at each node of the tree. Further developments of the Two Stage procedure face the computational efficiency problem. In fact, from a computational point of view, the growing procedure is crucial when dealing with very large data sets or when dealing with ensemble methods. At any node t the two stages can be defined as:

- **global selection**; one or more predictors are chosen as the most predictive for the response variable according to a given criterion; the selected predictors are used to generate the set of partitions or splits. In this stage an index needs to be defined to evaluate the Global Impurity Proportional Reduction (Global IPR) of the response variable Y at node t, due to the predictor X;

- **local selection**; the best partition is selected as the most predictive and discriminatory for the subgroups according to a given rule. In this stage one has to define an index as the Local Impurity Proportional Reduction (Local IPR) of the response Y due to the partition p generated by the predictor X

For classification trees the Global IPR is defined as τ index of Goodman and Kruskal

$$\tau_t(Y|X) = \frac{\sum_i \sum_j p_t^2(j|i)p_t(i) - \sum_j p_t^2(j)}{1 - \sum_j p_t^2(j)} \tag{1.3}$$

where $p_t(i)$, for $i = 1, \ldots, I$, is the proportion of cases in node t that have category i of X, and $P_t(j|i)$, for $j = 1, \ldots, J$, is the proportion of cases in the node t belonging to class j of Y given the i^{th} category of X. Note that the denominator in equation 1.3 is the Gini diversity index.
For regression trees, Global IPR can be defined as the Pearson's squared correlation η^2:

$$\eta^2_{Y|X}(t) = \frac{BSS_{Y|X}(t)}{TSS_Y(t)} \tag{1.4}$$

where SST is the total sum of squares of the numerical response variable Y and BSS is the between group sum of squares due to the predictor X.
In a similar way, the Local IPR for both classification and regression trees are defined as in equation 1.3 and 1.4, with the difference that in these cases indexes are computed between the response variable Y and the set of split s generated by the global IPR functions.
More precisely, for classification trees, at each node t of the splitting procedure, a split s of the I categories of X into two sub-groups (e.g.

$i \in l$ or $i \in r$), leads to the definition of a splitting variable X_s with two categories denoted by l and r. Local IPR is defined as

$$\tau_t(Y|s) = \frac{\sum_j p_l^2(j|tl)p_{tl} + \sum_j p_{tr}^2(j|r)p_{tr} - \sum_j p_t^2(j)}{1 - \sum_j p_t^2(j)} \tag{1.5}$$

whereas for regression trees it is

$$\eta^2_{Y|s}(t) = \frac{BSS_{Y|s}(t)}{TSS_Y(t)} \tag{1.6}$$

Two stage splitting criterion works as follow:

1. select the best predictor $X^*(t)$ at t node by maximising equation 1.3 or 1.4 for classification or regression problems respectively:

2. select the best split $s^*(t)$ at node t by maximizing equation 1.5 or 1.6 for all splits of $X^*(t)$ for classification or regression trees respectively

1.1.3 FAST splitting criterion

FAST algorithm (Fast Splitting for Splitting Tree) [84] provides a faster method to find the best split at each node when using CART methodology. As discussed in above section, when applying the two-stage criterion the best predictor could be found minimizing the Global Impurity Proportional Reduction factor due to any predictor X, then the Local Impurity Proportional Reduction factor determines the split with respect to all partitions derived from the best predictor.
Main issue of FAST is that the measure of Global IPR measure satisfies the following property:

$$\gamma(Y|X) \geq \gamma(Y|s) \tag{1.7}$$

in which γ is the generic Global IPR measure, and s is the set of split generated by X variable.
FAST algorithm consists in two step:

- computing Global IPR measure as in equation 1.3 or 1.4 for all variables belonging to the predictor matrix X and sort in decreasing order these measures;

- computing Local IPR measure as in equation 1.5 or 1.6 for the first previously ordered variable with maximum Global IPR. If Local IPR of this variable is higher than Global IPR of the second ordered X variable, stop the procedure, otherwise continue until inequality is satisfied.

The computational cost of FAST algorithm is really lower than the one of CART procedure, with the advantage that the final trees are exactly the same. In the example showed at the end of section 1.2.1 in the table 1.2, there is a set of six predictors, 2 nominal with 6 and 7 categories respectively, 3 ordinal with respectively 3, 5 and 4 categories and one binary variable. It was shown that CART procedure for each variable must examine each possible split to decide which one is the best. Considering the root node, CART technique has to compute (25-1)+(26-1)+2+1+4+3 = 104 splits. FAST algorithm computes at the beginning only six Global IPR measure (in this case there are only six predictors) and then only the local impurity reduction factor until inequality of the second step of the procedure is satisfied. In this small example, the number of computations made is 6+(25-1) = 30 (it is assumed that Global IPR measure relative to the second-best predictor is lower than the local impurity reduction factor obtained by the second one). The computational advantage of using FAST instead of CART is clear: one obtains the same tree-based structure with a great gain in terms of computational cost.

1.1.4 Stopping rules and assignment of the response classes/values to the terminal nodes

Once the rules for growing the tree has been defined, another set of rules to stop the building of the structure are needed. There is no unique rule to define the stopping of the procedure, but there are several rules used according the discretion of the researcher. Tree growing can be arrested considering a suitable combination of the following conditions:

- *Bound on the decrease in impurity.*
  A node is terminal if the reduction in impurity due to the further partition of the node is lower than a fixed threshold; a node should be splitted if their contribution to the total impurity reduction is significant;

- *Bound on the number of observations.*
  In general, can be useless to continue splitting nodes with a few number of individuals: sample size within-node should be "rational";

- Tree size.
  A further condition could be based on either the total number of terminal nodes or the number of levels of the tree to limit its expansion.

Once the tree has been built, terminal nodes must be associated with a response.
In the case of classification trees the assignment of a response to each terminal node is based on a simple majority rule. Specifically, node t is assigned to class j^* if the highest proportions of objects in node t belong to class j^* so that:

$$p(j^*|t) = \max_{j \in C} [p(j|t)]$$

In the case the response variable is numeric the response values for the object falling into a given terminal node t can be summarised by means of a synthetic measure; in general this is simply given by the mean, so that $\bar{Y}_t$ is assigned to node t where:

$$\bar{y}_t = \frac{1}{n(t)} \sum_{\mathbf{x}_n \in t} y_{i_t}$$

1.1.5 Pruning

Exploratory trees can be used to investigate the structure of data but they cannot be used in a straightforward way for induction purposes. For inductive purposes the question is: how large should be the tree? A very large tree might overfit the data, while a small tree may not be able to capture the important structure. Tree size is a tuning parameter governing the complexity of the model, and the optimal tree size should be adaptively chosen from the data. To choose the "honest" tree in terms of its size, Breiman *et al.* [16] defined the *minimal cost-complexity pruning.* Before proceeding with pruning description, the definition of an error measure of a tree structure is necessary.

- For classification trees, the error at the generic node t is defined as

  $$r(t) = \frac{1}{n_t} \sum_{i=1}^{n_t} (\hat{Y}_t \neq Y_i)$$

  where n_t is the size at t^{th} node, $\hat{Y}_t$ is the classification returned by the tree in the same node. The error rate of the overall tree is defined as

  $$R(T) = \sum_{h \in H_T} r(t)p(t)$$

where H_T is the set of all terminal nodes of the tree T, and $p(t)$ is the proportion of cases falling into the t^{th} terminal node.

- For regression trees the error rate is defined exactly as in equation 1.1, that is as the sum of TSS in the t^{th} node divided by the total sample size, whereas the prediction error of overall tree is defined as
$$RR(T) = \frac{R(T)}{R(t_1)}$$
where $R(t_1)$ is the error in the root node.

Pruning procedure works as follow: Let T_{max} be the maximum tree, let $\left|\tilde{T}\right|$ denote the set of all terminal nodes of T_{max}, that is its complexity. The cost-complexity measure is defined as

$$R_\alpha(T) = R(T) + \alpha \left|\tilde{T}\right| \tag{1.8}$$

where α is a non negative complexity parameter which "governs the tradeoff between tree size and its goodness of fit to the data" [61].
The idea is, for each α, find the subtree $T^*_\alpha \supseteq T_{max}$ to minimize $R_\alpha(T)$. When $\alpha = 0$ the solution is the full tree T_{max}, and the more α increases the more the size of the tree decreases.
The pruning procedure is the same for both classification and regression cases, so the attention can be focused on the classification problem without loss of generality. The cost complexity measure is defined for any internal node t and the branch T_t rooted at t as:

$$R_\alpha(t) = r(t)p(t) + \alpha$$
$$R_\alpha(T_t) = \sum_{h \in H_t} r(h)p(h) + \alpha \left|\tilde{T}_t\right|$$

where $R_{(t)}$ is the resubstitution error at node t, $p(t) = \frac{n(t)}{N}$ is the weight of node t given by the proportion of training cases falling in it and H_t

is the set of terminal nodes of the branch having cardinality $\left|\tilde{T}\right|$. The branch T_t will be kept as long as:

$$R_\alpha(t) > R_\alpha(T_t)$$

the error complexity of node t being higher than the error complexity of its branch. As α increases the two measures tends to became equal, this occurs for a critical value of α that can be found solving the above inequality:

$$\alpha = \frac{R(t) - R(T_t)}{\left|\tilde{T}_t\right| - 1} \tag{1.9}$$

so that α represents for any internal node t the cost due to the removal of any terminal node of the branch.
The pruning process produces a finite sequences of subtrees $\Omega = T_1 \subset T_2 \subset \ldots \subset T_{max}$, where T_1 is a tree constituted only by the root node. It can be proved [16] that the minimal cost-complexity pruning procedure produces the subtrees with the minimum error rate given the number of its terminal nodes. In other words, if T_α has five terminal nodes, there is no other subtree $T_s \subseteq T_{max}$ having five terminal nodes with smaller error ([16], pag. 71).
To validate a tree-based structure one has to consider its accuracy: the misclassification ratio or the prediction error. In both classification and regression cases an estimation of the error rate is needed. There are three possible ways to estimate it:

- *Resubstitution estimate*
  Resubstitution estimate is computed by using the same dataset used to build the tree. It is an optimistic estimate, therefore it is not used.

- *Test set estimate*
  If the sample size is sufficiently large, data can be randomly

splitted into two sub-samples (training sample and test sample). Then training sample is used to grow the tree-based structure and the test set is used to validate it.

- *Cross validation estimate*
  When sample size is not sufficiently large to be splitted into two sub-samples, one can use the cross-validation estimate. Data set is splitted into V sub-samples approximately of the same size, then V trees are built using the V^{th} sub-sample as test set and the other $V-1$ as training set. By averaging over the V test set estimates, finally the cross-validation estimate of the error rate is achieved.

A single final tree is then selected either as the one producing the smallest error estimate on an independent test set ($0-SErule$) or the one which error estimate is within one standard error of the minimum ($1-SErule$). Denoting by $R^{ts}(T)$ the test set error estimate associated with a generic tree T in the sequence Ω, according to the $0-SE$ rule the tree T^* will be selected if:

$$\mathrm{R}^{\mathrm{ts}}(\mathrm{T}*) = \min_{T \in S} \mathrm{R}^{\mathrm{ts}}(\mathrm{T})$$

whereas, if $1-SE$ rule is employed tree T^{**} will be selected if:

$$\mathrm{R}^{\mathrm{ts}}(\mathrm{T}^{**}) \leq \left[\mathrm{R}^{\mathrm{ts}}(\mathrm{T}^{*}) \pm \mathrm{SE}(\mathrm{R}^{\mathrm{ts}}(\mathrm{T}^{*}))\right]$$

1.2 Ensemble Methods

Ensemble methods are learning algorithm that construct a set of classifiers and then classify new data points by taking a vote of their predictions [32]. A necessary and sufficient condition for an ensemble of classifiers to work better than any single classifier is that the classifiers to be aggregate must be accurate (e.g. they must have an error

rate better than random choices) and diverse (e.g. the errors of the classifiers have to be unrelated).
There are several methods for constructing ensemble [32] (by enumerating the hypotheses or bayesian voting, by manipulating input features, by manipulating output targets), but the most popular ensemble methods work by manipulating the training examples.
These methods manipulate training examples through a re-sampling technique to generate multiple classifiers, then a learning algorithm is run several times with a different subset of training examples, as it can be seen in figure 1.2. The most famous ensembles belonging to this category are Bagging and Boosting.

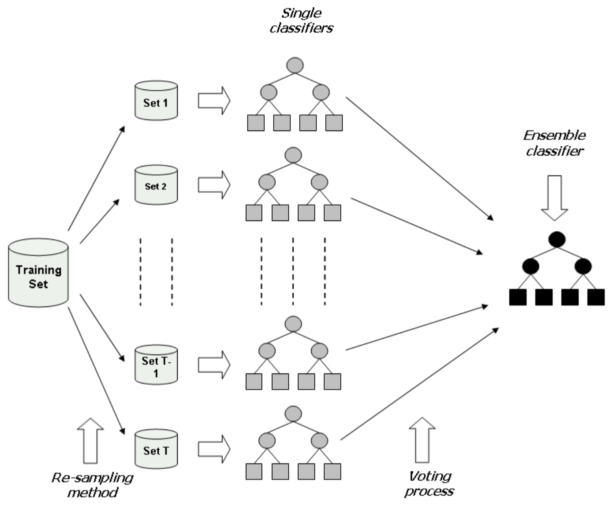


Figure 1.2: Ensemble methods working by manipulating training examples

1.2.1 Bagging

Bagging [13] is an acronym for Bootstrap Aggregating: it forms a set of classifiers that are combined by voting by generating replicated bootstrap [40] samples of the data. Table 1.3 shows the pseudo-code of Bagging algorithm.
Given a learning set $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$, the aim is to predict y using x as input by using a classifier $h(x, \mathcal{L})$. By using a sequence of t learning sets $\mathcal{L}_t$, with $t = 1, \ldots, T$, each consisting of n independent observations from the same distribution as in $\mathcal{L}$, goal is to get a better predictor than the single learning predictor set $h(x, \mathcal{L})$ coming from t bootstrap replications. The aggregating process is quite simple: if y is numerical the aggregated classifier is the average of each single classification over all the iterations of the procedure, if y is numerical the method of aggregating the classifiers is by voting.

```
Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and
$y_i \in R$ if numerical or $y_i \in \{1, \ldots, J\}$ if categorical.

• for $t = 1 : T$
    – generate a bootstrap replication $\mathcal{L}^B$ from $\mathcal{L}$
    – run a single classifier on $\mathcal{L}^B$
    – obtain the estimation $\hat{y}_i^t$ from the single classifier
• Output: final bagged classifier
  $\mathcal{H}(X) = \begin{cases} \text{aggregation by voting if } y_i \in \{1, \ldots, J\} \\ {}_{av}h\left(x, \mathcal{L}^B\right) \text{ if } y_i \in R \end{cases}$
```

Table 1.3: Bagging algorithm

Bagging works well for unstable procedures. Both classification and regression methods are unstable in the sense that small perturbations in their training sets or in construction may result in large changes in the constructed predictor [15]. In general, a classifier is unstable when it is affected by high variance, whereas a classifier is stable when it is affected by high bias [118, 129, 49]. So, Bagging is a method of variance reduction [52] and it works well with classification and regression trees because their are known as methods with high variance. Bagging returns worse performance than single classifiers when it is used with stable classifiers, e.g. with a stump [61]. As Breiman says [13]: *Bagging unstable classifiers usually improve them. Bagging stable classifiers is not a good idea.*

1.2.2 Boosting algortihms

Boosting is a general method for improving the accuracy of any given learning algorithm provided that single classifications are better than random choices. The main difference between Bagging and Boosting algorithms is that whereas Bagging uses the bootstrap as resampling method (that is, the probability of each individual to be included in the bootstrap training sample through the iterations is constant and equal to $1/n$, where n indicates the sample size), Boosting uses a *weighted* bootstrap, in the sense that the probability of each individual to be included in the boosted training sample is not constant, but it is weighted by the (good or bad) classification obtained by the learning sample. More precisely, starting from a uniform distribution of weights, these are increased for the i^{th} individual if he has been misclassified by the learning algorithm (or *weak learner*), otherwise these are decreased. This way, within the next iteration the probability of a misclassified instance to be included in the boosted training sample is higher than observations correctly classified, so the learning algorithm is forced to learn by its errors becoming a *strong learner*.

Therefore a weak learner is a supervised learning algorithm which returns a classification just slightly better than random choice, for example in the case of binary classification problems it must give back an error rate smaller than 50%.
Boosting has its roots in a theoretical framework for studying machine learning called the "PAC" learning model [121]. In brief, this theory states that a learning machine which is wrongly trained returns an incorrect prediction even if it is trained a lot of time, but with high probability a well trained learning machine will solve the classification problem after a certain number of tests. In other words, the machine has to be *Probably Approximately Correct*.
Several boosting algorithms are developed in the last years [100], such as polynomial-time boosting algorithm [101] and boosting-by-majority algorithm [46], but doubtless the most famous boosting algorithm is AdaBoost developed by Freund and Schapire in 1995 [47].

1.2.3 AdaBoost algorithms for classification and regression problems

Table 1.4 shows the pseudo-code of AdaBoost algorithm for binary classification problems. The algorithm takes as input a training set $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ in which $y_i = \{-1, +1\}$ and it calls a given weak learning algorithm repeatedly in a series of rounds $t, \ldots, T$. Main idea of the algorithm is to maintain a distribution of weights over $\mathcal{L}$. These weight are updated at each iteration t according to the weighted error occurred by the weak learner in the last iteration. Weak learner has to define a *weak hypothesis* $h_t : X \rightarrow \{-1, +1\}$ by which it is possible to compute the error $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$ (note that the error is computed over the distribution D on which the weak learner is trained). Subsequently the algorithm chooses an α parameter which indicates the importance of the weak hypothesis h_t to update the distribution D. The final boosted classifier, or *strong learner*, is the weighted majority vote of the T weak hypotheses weighted by α_t. AdaBoost is the acronym of Adaptive Boosting because it adapts to

---

Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i = \{-1, +1\}$

- initialize $D_1 = \frac{1}{n}$ for $i = \{1, \ldots, n\}$
- for $t = 1 : T$
  - train weak learner h_t using distribution D_t
  - obtain a weak hypotesys $h_t : X \to \{-1, +1\}$
  - compute the error $\epsilon_t = Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right]$
  - choose $\alpha_t = \dfrac{1}{2}\ln\left(\dfrac{1-\epsilon_t}{\epsilon_t}\right)$
  - update D distribution:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \epsilon^{-\alpha_t} \text{ if } h_t(i) = y_i \\ \epsilon^{\alpha_t} \text{ if } h_t(i) \neq y_i \end{cases}$$

$$= \frac{D_t(i)\exp\left(-\alpha_t y_i h_t\left(x_i\right)\right)}{Z_t}$$

where Z_t is a normalization factor

- Output: final boosted classifier:

$$\mathcal{H}(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t\left(x\right)\right)$$

Table 1.4: AdaBoost algorithm for binary response variable

the error rates of the individual weak hypotheses. The most basic theoretical property of AdaBoost concerns its ability to reduce the training error. It can be proved [47, 80] that the training error of the final hypothesis is at most

$$\frac{1}{n}\sum_{i=1}^{n}[H(x_i) \neq y_i] \leq \frac{1}{n}\sum_{i=1}^{n}\exp(-y_i\alpha_t h_t(x_i)) = \prod_{t=1}^{T} Z_t$$

By minimizing Z_t this limit error can be minimized, and this can be obtained by choosing the suitable α parameter. The expression $Z_t = \sum\limits_{i=1}^{n} D_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right)$ can be write as

$$Z_t = \sum_{i=1}^{n} D_t(i)\exp\left(-\alpha_t u_i\right) \tag{1.10}$$

where $u_i = y_i h_t(x_i) < 0$ if $y_i \neq h_t(x_i)$ and $u_i = y_i h_t(x_i) > 0$ if $y_i = h_t(x_i)$. If $Y \in \{-1, +1\}$, it follows that

$$\begin{aligned} Z_t = \sum_{i=1}^{n} D_t(i)\exp\left(-\alpha_t u_i\right) \leq \\ \leq \sum_{i=1}^{n} D_t(i)\left(\frac{1+u_i}{2}\exp(-\alpha_t u_i) + \frac{1-u_i}{2}\exp(\alpha_t u_i)\right) \end{aligned}$$

Recall that ϵ_t is the training error at t^{th} iteration, it can be indicated as $\epsilon_t = \frac{1-u_i}{2}$ and $(1-\epsilon_t) = 1 - \left(\frac{1-u_i}{2}\right) = \frac{1+u_i}{2}$. Equation 1.10 becomes

$$\begin{aligned} Z_t = \sum_{i=1}^{n} D_t(i) \exp\left(-\alpha_t u_i\right) \leq \\ \leq \sum_{i=1}^{n} D_t(i) \left((1-\varepsilon_t)\exp(-\alpha_t u_i) + \varepsilon_t \exp(\alpha_t u_i)\right) \end{aligned} \tag{1.11}$$

The last part of equation 1.11 can be wrote as

$$(1-\varepsilon_t)\exp(-\alpha_t) + \varepsilon_t \exp(\alpha_t) \tag{1.12}$$

so, to minimize Z_t one has to minimize expression 1.12 with respect to α and compute this parameter in that point, namely

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

By substituting this parameter in the expression 1.11 and by reducing, obtain

$$\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \left[\sqrt{4\varepsilon_t(1-\varepsilon_t)}\right] = \prod_{t=1}^{T} \left[2\sqrt{\varepsilon_t(1-\varepsilon_t)}\right] \tag{1.13}$$

If weak learner works better than random choice, $\epsilon_t = 0.5 - \gamma_t$ where γ_t is some positive parameter, and then $\gamma_t = 0.5 - \epsilon_t$. Equation 1.13 can be re-wrote as

$$\prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq \prod_{t=1}^{T} \exp\left(-2\gamma_t^2\right) = \exp\left(-2\sum_{t=1}^{T} \gamma_t^2\right) \tag{1.14}$$

in which the last term is the upper limit over training error of boosted classifier. Freund and Schapire [47] showed how to bound the generalization error of the final hypothesis in terms of its training error, the sample size n and the VC-dimension d of the weak hypothesis space (which is a standard measure of the complexity of a space of hypotheses [80]). They proved that the upper bound of the generalization error, which can be interpreted as the expected value of the test error [61], is at most [102]

$$P\left[\text{margin}_f\left(x,y\right) \leq \theta\right] + \tilde{O}\left(\sqrt{\frac{d}{n\theta^2}}\right)$$

for any $\theta > 0$, in which margin is a number in $[0,1]$ which is an indicator of the "confidence" of the classification.
For both multiclass and regression cases, main difference with the above described AdaBoost algorithm is about a suitable definition of the error and the computation of a loss function. Table 1.5 shows the pseudo-code of AdaBoost algorithm for multiclass classification problems. In a multiclass problem the condition that the error rate of the base classifier is less than 0.5 can be too restrictive. For this reason Freund and Schapire [47] introduced a pseudo-loss function of a confidence-rated classifier to be minimized in the boosting iterations instead of the error rate. The code in table 1.5 is the modified version of AdaBoost.M algorithm [47] as made by Eibl and Pfeiffer [38] and called AdaBoost.M1W.
Table 1.6 shows the AdaBoost code for regression problems. In this case, the error can be defined as a squared loss function, even if other loss functions, such as linear or exponential, could be used. In the regression case the final boosted classifier is obtained, in general, by averaging the single weak hypotheses through the iterations.
Code showed in the table 1.6 is the modification of AdaBoostR algorithm [47] made by Drucker [36, 56]. The aggregation process in this

```
Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and
$y_i = \{1, \ldots, J\}$

• initialize $D_1 = \frac{1}{n}$ for $i = \{1, \ldots, n\}$
• for $t = 1 : T$
    – train weak learner $h_t$ using distribution $D_t$
    – obtain a weak hypotesys $h_t : X \rightarrow \{1, \ldots, J\}$
    – compute the error $\epsilon_t = \sum_i D_t(i) I\left(h_t(x_i) \neq y_i\right)$
    – choose $\alpha_t = \ln\left(\frac{|J-1|\,(1-\epsilon_t)}{\epsilon_t}\right)$
    – update $D$ distribution:
          $$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t I(h_t(x_i)=y_i)}}{Z_t}$$
      where $Z_t$ is a normalization factor
• Output: final boosted classifier:
      $$\mathcal{H}(x) = arg\mathrm{max}_{y \in J}\left(\sum_{t=1}^{T} \alpha_t I\left(h_t(x) = y\right)\right)$$
```

Table 1.5: AdaBoost algorithm for multiclass classifiers

case is the weighted median.

Boosting is a method of bias reduction [61], therefore it can be used with stable classifiers (e.g. with a stump), but it works really good also in reducing variance of a classifier. Sometimes it can produce overfitting phenomenon, but it can occur when weak learner has a too high error rate or when the boosted training error reaches to zero too fast [47, 80, 102].

In general, ensemble methods allow to gain in prediction accuracy. When these are used in combination with tree-based models, if the goal is predicting as more as possible new observations, ensembles can help us to it achieve. If our goal is interpreting relationships among covariates, we could never use ensembles. As Breiman says [13], *what one loses, with the trees, is a simple and interpretable structure. What one gains is increased accuracy.*

Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i \in R$

- initialize $D_1(i) = \frac{1}{n}$
- for $t = 1 : T$
  - train weak learner h_t using distribution D_t
  - obtain a weak hypotesys $h_t : x \to y$
  - compute the quadratic loss function $L_t(i) = (y_i - h_t(x_i))^2$
  - compute an average loss $\epsilon_{D_t} = \sum_{i=1}^{n} D_t(i) L_t(i)$
  - set $\beta_t = \dfrac{\epsilon_{D_t}}{\max\limits_{1 \le i \le n} L_t(i) - \epsilon_{D_t}}$
  - set $w_k(i) = \dfrac{L_t(i)}{\max\limits_{1 \le i \le n} L_t(i)}$
  - set $g_t(i) = \beta_t^{1 - w_k(i)} D_t(i)$
  - update D distribution:

$$D_{t+1}(i) = \frac{g_t(i)}{\sum_i g_t(i)}$$

- Output: final boosted classifier:

$$\mathcal{H}(x) = inf \left\{ y \in Y : \sum_{t : h_t \le y} \log\left(\frac{1}{\beta_k}\right) \ge \frac{1}{2} \sum_t \log\left(\frac{1}{\beta_t}\right) \right\}$$

Table 1.6: AdaBoost algorithm for regression problems

Chapter 2

Data Editing

2.1 Data pre-processing

Data pre-processing describes any type of processing performed on raw data to prepare it for another processing procedure. Commonly used as a preliminary data mining practice, data pre-processing transforms the data into a format that will be more easily and effectively processed for the user's purpose. There are several tools and methods used for pre-processing, including Data editing and Data fusion.

2.1.1 Data Editing

Data editing is the process by which data collected in some way (a statistical survey for example) is examined for errors with the help of software. Winkler [128] defined Statistical Data Editing (SDE) as those methods that can be used to edit (i.e., clean-up) and impute (fill-in) missing or contradictory data. SDE can be used in all phases of survey process. These phases include frame development, form design, proposed analytic purposes for which the data are collected, and quality assurance. The main goal of SDE might be improved procedures and greater automation to enhance the ability of survey managers and analysts to provide published estimates and micro-data.

Editing begins with the specification of a set of **edits**. They are conditions which should be met by the data. For example, by considering a socio-economic survey, possible edits are:

- $AGE \rightarrow$ integer number between 0 and 120
- $MARITAL\ STATUS \rightarrow$ one of these categories: $SINGLE$, $MARRIED$, $DIVORCED$, $WIDOWED$
- if $AGE < 15$ then $MARITAL\ STATUS$ must be $SINGLE$

These examples of edits involve data of single respondents, so that these are **micro-edits**. Edits which involve data of several respondents are **macro-edits**.
A given set of edits is not necessarily correct. It may omit important edits or it may contain edits which are conceptually wrong, too restrictive or logically inconsistent. The extent of these problems is reduced by having *subject-matter experts* specify the edits of a survey. The problem is not eliminated however, because many surveys conducted in practice involve an huge number of items and require hundreds of edits, which makes their specification a very demanding task. As a check, a proposed set of edits is applied on test data with known errors before application on real data. Missing edits or logically inconsistent edits however, may not be detected. Problems in the edits, if discovered during the actual editing or even after it, cause editing to start again after their correction, leading to delays and incurring larger costs than anticipated.

The editing literature does not contain many relevant suggestions, nevertheless SDE can be broadly subdivided in two subcategories:

1. General methods
2. Fellegi-Holt methods and systems

The first category includes traditional methods. These methods comprise *If-Then-Else* rules for detecting contradictory information and various ways of imputing values or variables to replace the contradictory ones. These rules may not be straightforward to develop and may be difficult to write into computer code. Indeed, slight changes in the survey form and edit rules can cause the need to rewrite and debug thousands of lines of code. For that reason other methods have been studied and developed, such as the one based on Exploratory Data Analysis (EDA) as point and click method tolls for finding erroneous data [31].

The second category is characterized by the Fellegi-Holt (FH) model [45] of editing. In brief the method provides a way of automatically generating all edits implied by a set of explicitly defined edits. This reveals all logical inconsistencies in the edits and on the other hand guarantees that data records which fail edits will be made to satisfy them with the minimum possible modification. FH methods are so appealing because most of the *If-Then-Else* type of edits can be put in tables which are straightforward to modify and update. Because the source code does not need any updating, it is possible to create a FH system for editing which can be developed and maintained for different surveys by non-programmers such as subject matter specialists. A disadvantage, that prohibits the application of the method as intended is that for reasonably-sized surveys it has great computing demands, even by today standards. Moreover, the method mainly applies to categorical data.
Other than that, the editing literature has concentrated on improving the performance of editing in practical applications, mainly on two grounds: speed of implementation and number of different types of

edits that can be accommodated by a single piece of software.

Recently both a new step in the edit specification process and a new method for automatically specifying the functional form of edits have been developed [88]. The new step consists in formally describing the structure of the phenomenon studied by a given survey (definition of an *abstract data model*), the new method follows a new strategy that considers first tree-based models to derive edits from clean data and then a statistical criterion to validate new incoming data. This strategy is called **TREEVAL**.

2.1.2 TREEVAL strategy

Conceptually each survey, by registering data about distinct independent occurrences of a particular real-world phenomenon, results in a set of corresponding independent observations, materialised collectively in the form of a data set. It is worth stating well in advance, that the abstract data model of the survey directly reflects the structural model of the real-world system. Hence, by departing from the usual concept of a survey data set as a mere collection of data, abstract model to specify edits can be considered as a system of information, whose logical structure directly reflects the structure of the specific real-world system under investigation. It follows that edits are not just logical expressions involving variables, but they directly reflect the real constraints on the relationships among the components of this real-world system and on their properties. So, edits derive from the model of the system to be investigated, and can be identified by means of formal analysis of this model.

Tree-based methods can be fruitfully used in data editing to automatically derive probabilistic edits. In fact, all edits are logical statements about the relationship among the involved variables; therefore,

by building trees on clean data these statements can be automatically identified. This way, no subject matter experts will be required to specify edits, but these will be developed automatically by a software that implements recursive partitioning algorithm. For example, let X_i, X_j and X_k be three variables and assume that the relationship among them goes from X_i and X_j to X_k. Then, X_k will play the role of response variable in the binary segmentation procedure whereas X_i, X_j are the explanatory variables. The paths that lead to any terminal node of the resulting tree state that specific values of X_k can arise only from given combinations of values of X_i and X_j. The inspection of these connections can suggest suitable edits which can be jointly used with those provided by the abstract data model.
Figure 2.1 shows the TREEVAL procedure for automated derivation of edits [88]. By referring to surveys which are repeated periodically,

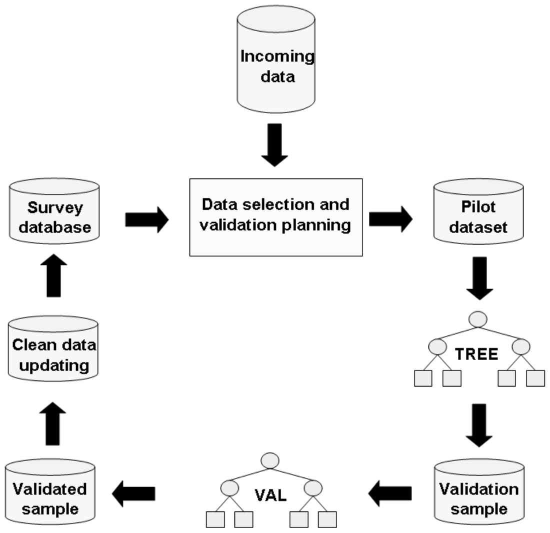


Figure 2.1: TREEVAL strategy

the *survey database* is the database storing the data derived for previous similar surveys. It is assumed to contain **clean data** because it refers to data that were validated in the past. The *incoming data* is the data that must be validated before being included into the survey database. The validation process is not applied on the data of the whole survey simultaneously but on subsets of cases or variables which are selected according to a stratifying variable to define homogeneous strata in the data set. This allows to simplify the analysis as well as to validate cases which are in some way heterogeneous. The steps of TREEVAL procedures are:

- Collect data from survey database to derive edits.
  To this aim a *Data Selection and Validation Planning* is performed. Practically, once the set of variables to be validated has been defined, the relative cases are selected from the *survey database* in order to form the *pilot dataset*, which will be used to derive edits.

- TREE phase.
  Pilot dataset is assumed to be formed of N objects and P variables. The aim is deriving edits for each of the variables composing the pilot dataset. This goal can be achieved by growing P tree-based structures in such a way that each of the variables of this dataset is considered in turn as a response variable. Each final tree is selected by growing the maximal tree and pruning it according to the test sample criterion or the cross-validation. As a result, for each terminal node of each tree an edit is derived considering the different paths and their associated variables. The ratio between the within group standard deviation of the response variable in the terminal node over the within group standard deviation of the response variable in the root node is considered as a gain measure for each edit. That is, the gain for

each edit is

$$\text{gain}(edit) = \frac{\hat{\sigma}_Y(h)}{\hat{\sigma}_Y(t_1)}$$

where $h = 1, \ldots, H$ is the set of terminal nodes and t_1 is the root node of the tree. This gain measure is used to order the production rules and thus the edits on the basis of their predictability power. Three groups of edits can be identified using suitable thresholds, for example *strong edits* if $\text{gain}(edit) < k$, *middle edits* if $\leq k\ \text{gain}(edit) \leq t$, *weak edits* if $\text{gain}(edit) > t$.

- VAL phase
  Once a set of edits have been constructed and selected automatically, it can be used to validate the data. So, in the VAL (validation) phase the set of previously identified edits is applied on the new data. On the *incoming data* the same data selection criteria used when deriving the pilot dataset in order to derive the *validation sample* are applied. Validation sample is then the partition of the incoming data to be validated. Briefly, cases of validation sample are passed down the P trees and the difference between the imputed and the observed values are measured. This measure changes according to the nature of the variable which, at turn, plays the role of response variable. If a case of the validation sample is not consistent with most of the validation rules, this case is suspected to be an error and it can be corrected according to the specification of the relative validation rules or can be deleted from the database. Finally, the cleaned dataset derived from the validation sample defines the *validated sample.*

- Clean data updating.
  The validation sample now contains data that passed successfully the validation step, so that they can used to upgrade the

survey database. In other words it contains clean data and it is periodically updated as the validation process goes on. Therefore, new validated cases concur to validate not yet validated ones.

2.1.3 Missing Data Imputation

Missing data imputation is a really remarkable step in the framework of data editing. Missing or incomplete data are a serious problem in many fields of research because it can lead to bias and inefficiency in estimating the quantities of interest. The relevance of this problem is strictly proportional to the dimensionality of the collected data. Particularly, in data mining applications, a substantial proportion of the data may be missing and predictions might be made for instances with missing inputs. Specially in the case in which data coming from different self-updating repositories, and final data sets are really huge, it is not unusual to have to work with a lot of missing data. Therefore their imputation is a fundamental preliminary step, the goodness of final validation of the rules depending on it. In the framework of missing data imputation, it is worthwhile to distinguish between missing data completely at random and missing data at random [72]. More precisely, data is missing completely at random (MCAR) when the probability that an observation is missing is unrelated to the value of the variable or to the value of any other variables. Instead, data can be considered as missing at random (MAR) if the data meets the requirement that missingness does not depend on the value of the variable after controlling for another variable. While the MCAR condition means that the distribution of observed and missing data is indistinguishable, the MAR condition states that the distributions differ but missing data points can be explained (and therefore predicted) by other observed variables. The last condition is the more relevant one in statistics, because it requires a model-based imputation so that

the missing value can be understood as the sum of the model function and the error term.
Chapter three of the present work is dedicated to the framework of missing data imputation and, in particular, to the use of ensemble tree-based models as unique tools to proceed to the imputation

2.2 Data Fusion

Data fusion aims at matching two already held surveys in order to make possible transferring part of information from the first survey to the second one. Hence, it allows to treat the data coming from the two distinct surveys as whole.
Data Fusion methods deal with two data sets, the first containing the information related to a set of $p+q$ variables observed on n_0 subjects, the second consisting of p variables observed on a sample of n_1 subjects. It is assumed that the set of the p variables is common, i.e. observed on both samples, while the set of q variables are specific, i.e. not included in the both surveys. Let $\mathbf{X}$, $\mathbf{X}_0$ and $\mathbf{X}_1$ denote respectively the generic data matrix of the p common variables, the same one if referred to the donors matrix or if referred to the receiver matrix. Furthermore, let $\mathbf{Y}$ be the matrix of q specific variables of the reference survey.
The aim is to determine the unobserved values of the q variables. In other word, $\mathbf{Y}_1$ is considered as a missing data matrix to be imputed. As a consequence, the Data Fusion can be considered as a particular case of data imputation framework, with the difference that in this case a group of instances is missing as they have not been collected.

According our opinion Data Fusion can be included in Data Editing schema, Data Editing being a process with which one "prepares" data to be analysed in the best way in a second time. On this extent Data

Fusion, which has the aim to complete a data matrix by extending to them results coming from another dataset, can be considered as a method to ”clean” data and, consequently, to be ready to anlyse them. Indeed, following Saporta’s words [97], *in data fusion, the goal is to obtain a single data base where all variables have been completed for the union of units. The resulting base may be analysed afterwards with data mining tools.*

in any case, this is not the right place for discuss if Data Fusion belongs in the framework of Data Editing or not, doubtless it could be an arguable point of view. According to our opinion it can be seen as belonging of data pre-processing category, being Data Fusion a very special case of missing data imputation.
Chapter four is about Data Fusion framework and, in particular, it deals with tree-based methods as complete tools to proceed to the ”fusion”.

Chapter 3

Boosted Incremental Non Parametric Imputation of Missing Data

3.1 Introduction

In this chapter, a general tree-based methodology for missing data imputation as well as specific algorithms to obtain the final estimates are provided. Missing or incomplete data are a serious problem in many fields of research because they can lead to bias and inefficiency in estimating the quantities of interest. In data mining applications a substantial proportion of the data may be missing and predictions might be made for instances with missing inputs. In recent years, several approaches for missing data imputation have been presented in the literature. Main reference in the field is the Little and Rubin book [72] on statistical analysis with missing data. An important feature which characterize an incomplete data set is the missing data mechanism which takes into account the process generating missing

values. In most situations, a common way for dealing with missing data is to reject records with missing values and restrict the attention to the completely observed records. This approach is based on the restrictive assumption that missing data are Missing Completely At Random (MCAR), i.e., that the missing-ness mechanism does not depend on the value of observed or missing attributes. This assumption rarely holds, however discarding the records with missing data is not an option [98].
An alternative and weaker version of the MCAR assumption is the Missing at Random (MAR) condition. Under a MAR process, the missigness depends on observed data but not on missing data themselves. While the MCAR condition means that the distributions of observed and missing data are indistinguishable, the MAR condition states that the distributions differ but missing data points can be explained (and therefore predicted) by other observed variables.
Last condition requires a model-based imputation for that the missing value can be understood as the sum of the model function and the error term. Classical approaches are linear regression [73], logistic regression [120], generalized linear models [69], whereas more recent approaches are nonparametric regression [19] and tree-based models [112]. Parametric and semi-parametric approaches can be unsatisfactory for nonlinear data yielding to biased estimates if model misspecification occurs. As an alternative, tree-based models do not require the specification of a model structure, deal with numerical and categorical inputs, consider conditional interactions among variables so that they can be used to derive simple imputation rules. Automatic tree-based procedures have been already considered for data validation [88] as well as for data imputation [112].

3.2 Missing data mechanism

Missing data mechanism was formalized by Rubin [96] through the idea of treating a missing data indicator matrix as random variable and assigning them a distribution.
Let $Y = (y_{ij})$ be the full data matrix, and let $M = (M_{ij})$ be the missing data indicator matrix, with $(M_{ij}) = 1$ if the corresponding (y_{ij}) is missing, and $(M_{ij}) = 0$ otherwise.
Missing data mechanism is characterized by the conditional distribution of M given Y, for example $f(M|Y, \psi)$, where ψ denotes unknown parameters.
If

$$f(M|Y, \psi) = f(M|\psi) \text{ for all } Y, \psi \tag{3.1}$$

that is, if missingness does not depend on observed and/or missing values of Y, data are called Missing Completely at Random (**MCAR**). When data are MCAR, there is the highest level of randomness. In other words, it occurs when the probability of an instance having a missing value for an attribute does not depend on either the known values or the missing data.
Let Y_o be the complete part of Y matrix, and let Y_M the part of Y matrix bearing missing data.
If

$$f(M|Y, \psi) = f(M|Y_o, \psi) \text{ for all } Y_o, \psi \tag{3.2}$$

that is, if missingness depends only on Y_o and not on Y_M, data are called Missing at Random (**MAR**). In other words, data are MAR when the probability of an instance having a missing value for an attribute may depend on the known values, but not on the value of the missing data itself;
If the distribution of M depends on the missing values Y, namely if

$$f(M|Y, \psi) = f(M|Y_o, Y_M, \psi) \text{ for all } \psi \tag{3.3}$$

data are called Not Missing at Random (**NMAR**). In other words, data are NMAR when the probability of an instance having a missing value for an attribute could depend on the value of that attribute.

3.3 Missing data treatment

In general, missing data treatment methods can be divided into the following categories [72]:

1. Ignoring and deleting data. This category can be splitted in two sub-categories according the way to discard data with missing values:

    - Listwise deletion, or complete case analysis. It simply consists in deleting all cases with missing data.
    - Discarding instances and/or attributes. This method consists of determining the extent of missing data on each instance and attribute, and delete the instances and/or attributes with high levels of missing data. Before deleting any attribute, it is necessary to evaluate its relevance to the analysis.

2. Available-case methods. Most famous technique belonging to this category is the *pairwise delection*, which works by deleting information only from those statistics that need the information.

3. Imputation-based procedures. Data imputation is a class of procedures that aims to fill in the missing values with estimated ones. The objective is to assume known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. There are a variety of methodologies which have been developed in the literature for data imputation. They can be grouped into two class of models:

- Non-model based imputation procedures, such as unconditional mean imputation, which allows to use the mean value of a variable in place of missing values for the same variable, and conditional mean imputation (or Buck's method) which computes different linear regressions for each pattern of missing data.

- Model-based imputation procedures. These procedures can be further spitted in two categories:
  Implicit models.
  Implicit models are based on implicit assumptions such as the proximity between individuals belonging to the data set. Most common techniques in this category are the *hot deck imputation*, which involves substituting individual values drawn from similar respondent units, *cold deck imputation*, which replaces a missing value by a constant value from an external source, *substitution method*, which replaces non-responding units with alternative units not selected into the sample in general at the field-work stage of the survey, and *composite methods*, which combines ideas from different methods (e.g. hot deck and regression).
  Explicit models.
  Imputations are based on a formal statistical model of the predictive distribution of missing data, so the assumptions are explicit. These procedures can be further described in terms of two approaches:

  - *Parametric approach*, which includes imputation through linear regression, the logistic regression, the imputation via maximum likelihood methods (e.g. with EM algorithm), multiple imputation methods.
  - *Non parametric approach*, which includes the imputation via non parametric regression [19] and through

tree-based models [112].

This chapter focuses on non parametric model-based imputation, in particular by using classification and regression trees combined with ensemble procedures.

3.4 The incremetal imputation approach

The incremental imputation methodology is based on the assumption that data are missing at random so that the mechanism resulting in its omission is independent of its unobserved value. Furthermore, missing values often occur for more variables. Given a variable for which data needs to be imputed, and a set of other observed variables, the method considers the former as response and the latter as predictors in a tree-based estimation procedure. Indeed, terminal nodes of the tree are internally homogeneous with respect to the response variable providing candidate imputation values for this variable. In order to deal with imputations in many variables the incremental approach is based on a suitably defined pre-processing schema, generating a lexicographic ordering of different missing values [112]. The idea is to rank missing values that occur in different variables and deals with these incrementally, i.e, augmenting the data by the previously imputed values in records according to the defined order.
Let X be the $n \times k$ original data matrix, where there are $d < k$ completely observed variables and $m < n$ complete records.
Define the matrix R to be the indicator matrix with ij-th entry 1 if x_{ij} is missing and zero otherwise. Summing up over the rows of R yields to the p-dimensional row vector c which includes the total number of missing values for each variable. Summing up over the columns of R yields to the n-dimensional column vector r which includes the total number of missing values for each record. A two-way re-arrangement of

X is performed, one with respect to the columns and one with respect to the rows. This process allows to define a lexicographic ordering (Cover and Thomas, 1991; Keller and Ullman, 2001) of the data that matches the ordering by the number of missing values occurring in each row and column of X. In particular, the re-arrangement satisfies the following missing data ranking conditions:

- $r_1 \leq r_2 \leq \ldots \leq r_n$ for the rows, with r_i the entry for r

- $c_1 \leq c_2 \leq \ldots \leq c_n$ for the columns, with c_j the entry for c.

As in the X data matrix there are $d < k$ completely observed variables and $m < n$ complete records, it holds that $r_i = 0$ for $i = 1, ..., m$ and $c_j = 0$ for $j = 1, ..., k$.
Let Z be the re-arranged data matrix; it can be partitioned in four sub-matrices: $A_{m,d}$, $B_{m,k-d}$, $C_{n-m,d}$, $D_{n-m,k-d}$, where only sub-matrix D contains missing values while the other three blocks are completely observed in both rows and columns (see figure 3.1).

The general idea is to use the complete part of Z to impute - via decision trees - the uncomplete one D, and on the basis of the lexicographic ordering the imputed values enter in the complete part for the imputation of the remaining missing values. Applying the tree-based procedure, each variable bearing missing values is considered as response variable at turn, whereas the remaining variables presenting either no missing values or imputed values are used as predictors. The imputation is incremental because, as it goes on, more and more information is added to the data matrix, block C iteratively is included in the block A and block D iteratively becomes block B. As a result, cross-validated trees are used to impute data and the algorithm performs an incremental imputation of each single data at time.

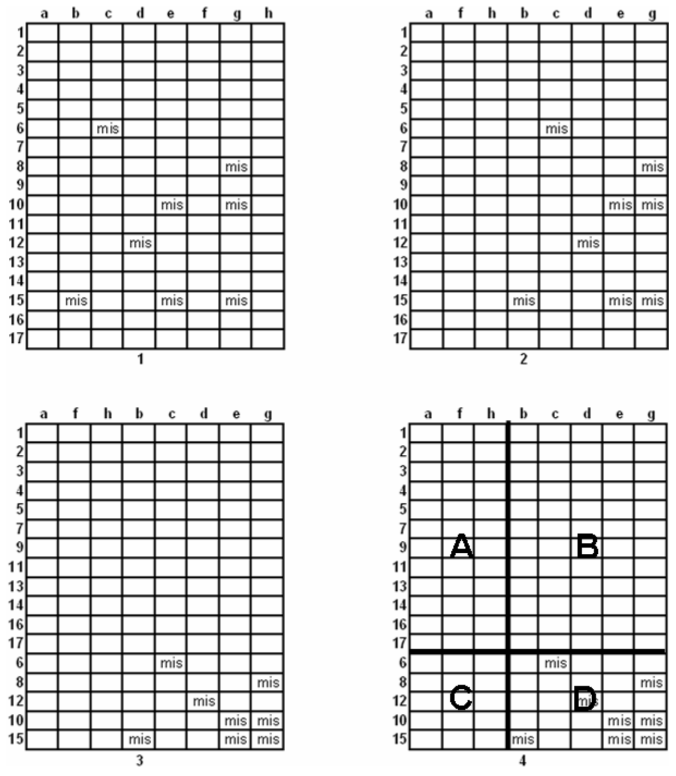


Figure 3.1: The Incremental imputation process.
1: The original data matrix X; 2: Re-arrangement according the number of missing-ness in the columns 3: Re-arrangement according the number of missing-ness in the rows 4: The final Z matrix partitioned in four blocks. D block contains missing data, whereas A, B and C blocks are all complete

3.5 Boosted Incremental Non Parametric Imputation

The above-mentioned incremental imputation approach is revised in this section by considering two new concepts:

- the use of ensemble methods (in place of cross-validation) should provide more robust estimates;
- the incremental imputation of each variable at time (instead of each single data at time) allows a more efficient algorithm, thus reducing the computational cost of the overall procedure.

For a $n \times k$ data matrix X a lexicographic ordering of the variables is defined as the k-dimensional vector $l = \left[l_{(1)}, \ldots, l_{(j)}, \ldots, l_{(k)}\right]$ such that $l_{(j)}$ points the column of the variable that is at the j-th position in the increasing order of all variables in terms of the number of missing values. It is assumed that at least the first ordered variable presents no missing values. An imputation algorithm by an ensemble of trees is considered using as *weak learners* a *stump* (a tree with only two terminal nodes) for imputation of a qualitative binary variable, and fast trees [84, 83] for the other cases. As ensemble methods, Boosting algorithms are used [47, 100, 102, 80, 56], and they are preferred to Bagging [13] specially when a *stump* is used as learning algorithm because it is known as a classifier with high bias [61] and, as it is well known, Bagging works with unstable classifiers. Figure 3.2 shows an example of the main steps of the basic imputation algorithm. It is worth noting that numbers associated to the columns correspond to the values of the lexicographic order vector $l_{(j)}$.

Let Y a $n \times k$ matrix bearing missing data where y_k is the k-th variable of Y.

- let R be the number of variable of Y containing missing data;

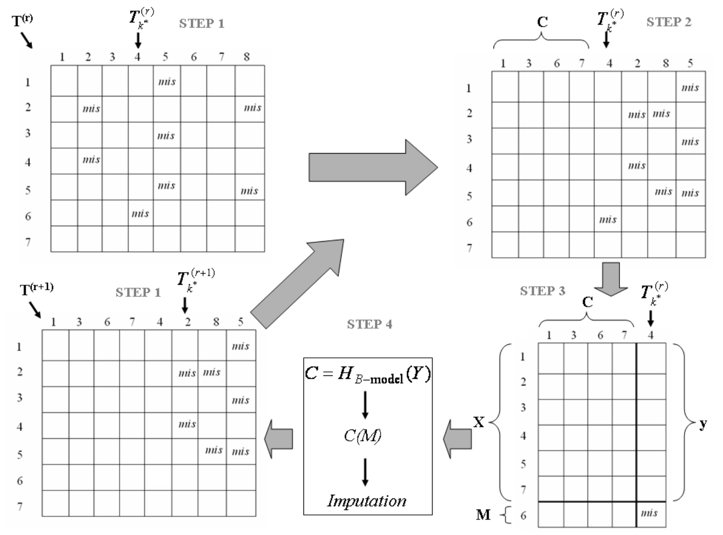


Figure 3.2: Boosted Incremental Non Parametric Imputation Algorithm

- for $r = 1, \dots, R$
  - find $y^r_{k^*}$ as the variable with the smallest number of missing data, where $k^* : \#mis_{k^*} \leq mis_k$, for $k = 1, 2, \dots, k$ and $\#mis_k > 0$;
  - sort columns such that the first p variables are complete and the $p + 1 - th$ is $y^r_{k^*}$;
  - sort rows such that the first l rows are complete and the remaining $n - l$ are missing in the $p + 1 - th$ column;
  - let $\mathcal{L}^{(r)} = \{y^{(r)}_{nk^*}, \mathbf{x}^{(r)}_n = (x_{n1}, \dots, x_{np})^{'}\}$ for $n = 1, \dots, l$ be the learning sample.
    Use a tree-based classifier as weak learner for v-fold AdaBoost iterations to impute the $n - l$ missing data in variable $y^r_{k^*}$. Use a STUMP if $y^r_{k^*}$ is a binary variable, use a

FAST classification tree if $y^r_{k^*}$ is a multi-class categorical variable, and use a FAST regression tree if $y^r_{k^*}$ is a numerical variable.

- output: all missing data are imputed

3.6 Simulation study

To test how BINPI algorithm works, a simulation study has been designed. The basic assumption is that missing data are generated according to a missing at random **MAR** schema so that a dependence relationship structure among variables is defined. The simulation setting consists in varying both the number of missing values in the uncompleted variables and the type of relationships between the uncompleted variables and the complete ones. In each setting, covariates are uniformly distributed in $[0, 10]$ and values are missing with conditional probability:

$$\Psi = \left[1 + exp\left(\alpha + \beta\right)\right]^{-1} \tag{3.4}$$

$BINPI$ algorithm has been evaluated with respect to standard methods such as Unconditional Mean Imputation (UMI) and Parametric Imputation (PI). A further comparison takes account of Incremental Non Parametric Imputation ($INPI$). In the following, the case of missing values generation in nominal covariates (i.e., nominal response case) is treated separately from the case of missing values generation in numerical covariates (i.e., numerical missing case).

3.6.1 Binary missing case

In each simulation setting, two different cases are considered, linear as well as non-linear relationships. Five data structures have been considered for the nominal response case. The variables under imputation are simulated according to the binomial distribution. In the *simulation 1*, *simulation 2* and *simulation 3* the parameters characterizing the distribution of the missing values are expressed as a linear combination of some of the covariates.

Simulation 1:

$$Y_1 \sim Bin\left(n, \frac{2+0.35(X_1+X_2)}{10}\right),\ m_1 = \{1 + exp\left[-3 + 0.5\left(X_1 + X_2\right)\right]\}^{-1};$$
$$Y_2 \sim Bin\left(n, \frac{4+0.35(X_2-X_3)}{10}\right),\ m_2 = \{1 + exp\left[-3 + 0.5\left(X_2 - X_3\right)\right]\}^{-1}.$$

Simulation 2 (*to simulation 1 the following variable is added up*):

$$Y_3 \sim Bin\left(n, \frac{3+0.35(X_3+X_4)}{10}\right),\ m_3 = \{1 + exp\left[-3 + 0.5\left(X_3 + X_4\right)\right]\}^{-1}.$$

Simulation 3 (*with respect to simulation 1 the second variable is replaced by*):

$$Y_2 \sim Bin\left(n, \frac{5+0.35(X_2-X_3)}{10}\right),\ m_3 = \{1 + exp\left[-3 + 0.5\left(X_3 + X_4\right)\right]\}^{-1}.$$

In the *simulations 4* and *5* the parameters characterizing the distribution of the missing values depend on some covariates in a non linear way.

Simulation 4:

$$Y_1 \sim Bin\left(n, \left|\sin\left(0.3X_1 + 0.9X_2\right)\right|\right)$$
$$m_1 = \{1 + exp\left[1.5 + 0.5\left(X_1 + X_2\right)\right]\}^{-1};$$

$$Y_2 \sim Bin\left(n, \left|\sin\left(0.9X_2 + 0.3X_3\right)\right|\right)$$
$$m_2 = \{1 + exp\left[1.5 + 5\left(0.3X_2 + 0.9X_3\right)\right]\}^{-1}.$$

Simulation 5 (*to simulation 4 the following variable is added up*):

$Y_3 \sim Bin\,(n, |\sin(0.5X_3 + 0.5X_4)|)$
$m_3 = \{1 + exp\,[1.5 + 0.5\,(0.5X_3 + 0.5X_4)]\}^{-1}$.

# errors	Y_1	Y_2	π_1	π_2
UMI	0	81	0.2130	0.1620
INPI	2	0	0.2150	0.2430
BINPI	0	0	0.2130	0.2430
TRUE			0.2130	0.2430
# missings	203	81		

Table 3.1: Simulation 1: main results

Tables from 3.1 to 3.5 show the results of the five simulations concerning the case of missing data presented in dummy variables. Each simulation was performed with two goals: estimating the expected value parameter of each binomial distribution (to be compared with the true value) as well as calculating the number of uncorrect imputations in each variables. Doesn't matter that an estimation of the probability of success near to the true value does not imply necessarily a correct imputation. The empirical evidence demonstrates the overall good performance of $BINPI$ over $INPI$ in terms of accuracy. This can be justified with two properties of $BINPI$:

- by definition a larger sample is used to build up the classifier;
- a more accurate learner is considered.

Finally, $BINPI$ provides a variable imputation (instead of a single data imputation) yielding to a computationally more efficient procedure that can be recommended in the analysis of large data sets such as in statistical offices surveys.

# errors	Y_1	Y_2	Y_3	π_1	π_2	π_3
UMI	0	80	804	0.2120	0.1980	0.1550
INPI	1	0	432	0.2130	0.2780	0.3730
BINPI	0	0	84	0.2120	0.2780	0.7350
TRUE				0.2120	0.2780	0.8190
# missings	169	80	808			

Table 3.2: Simulation 2: main results

# errors	Y_1	Y_2	Y_3	π_1	π_2	π_3
UMI	164	78	191	0.6260	0.4350	0.6470
INPI	2	1	1	0.4640	0.5120	0.4570
BINPI	0	0	0	0.4620	0.5130	0.4560
TRUE				0.4620	0.5130	0.4560
# missings	169	78	191			

Table 3.3: Simulation 3: main results

3.6.2 Numerical missing case

Two different data structures for the numerical response case have been generated. The variables under imputation are obtained according to the normal distribution, and data were missing according to the conditional probability as in equation 3.4.
Simulation 6 presented missing values in two variables, whereas in *simulation 7* missing data occur in three covariates.

Simulation 6:

$Y_1 \sim N\left(X_1 - X_2^2, exp\left(0.3X_1 + 0.1X_2\right)\right)$
$m_1 = \left\{1 + exp\left[-1 + 0.5\left(X_1 + X_2\right)\right]\right\}^{-1}$.

# errors	Y_1	Y_2	π_1	π_2
UMI	25	17	0.1630	0.1710
INPI	45	95	0.1870	0.2630
BINPI	24	94	0.1640	0.2620
TRUE			0.1760	0.1880
# missings	151	138		

Table 3.4: Simulation 4: main results

# errors	Y_1	Y_2	Y_3	π_1	π_2	π_3
UMI	76	86	77	0.6070	0.6320	0.6190
INPI	70	86	84	0.4510	0.6320	0.5216
BINPI	62	72	66	0.4670	0.6200	0.5600
TRUE				0.5310	0.5460	0.5420
# missings	180	170	180			

Table 3.5: Simulation 5: main results

$$Y_2 \sim N\left(X_3 - X_4^2, exp\left(-1 + 0.5\left(0.3X_3 + 0.1X_4\right)\right)\right)$$
$$m_2 = \left\{1 + exp\left[-1 + 0.5\left(X_3 + X_4\right)\right]\right\}^{-1}.$$

Simulation 7 (*to simulation 6 the following variable is added up*):

$$Y_3 \sim N\left(X_5 - X_6^2, exp\left(-1 + 0.5\left(0.2X_5 + 0.1X_6\right)\right)\right)$$
$$m_3 = \left\{1 + exp\left[-1 + 0.5\left(X_5 + X_6\right)\right]\right\}^{-1}.$$

Tables 3.6 and 3.7 show that *BINPI* algorithm works better than other techniques. The stump as learning algorithm, even if performs better than UMI and PI, is not suitable for numerical case. AdaBoost

	μ_1	μ_2	σ_1	σ_2
True	-28.5621	-27.6313	30.4572	29.8676
UMI	-37.3740	-36.2881	23.5282	24.0916
PI	-24.1569	-23.4112	36.0030	35.1718
INPI	-30.6277	-29.9536	30.4772	30.2005
BINPI	-29.2998	-27.9536	29.1428	20.0064
BINPI *stump*	-30.1244	-28.6836	28.8604	28.7842

Table 3.6: Simulation 6: main results

algorithm used in the numerical case is the one as described by Gey and Poggi [56]. The use of boosting improve imputation performance.

	μ_1	μ_2	μ_3	σ_1	σ_2	σ_3
True	-28.7185	-28.8756	-28.8333	30.2523	30.1449	30.0950
UMI	-37.3008	-38.0560	-37.9888	21.3271	22.8859	24.8487
PI	-24.6298	-24.5213	-24.5822	35.3920	35.6813	35.5709
INPI	-30.2286	-29.7767	-29.8391	30.6879	30.1415	30.3985
BINPI	-29.0670	-29.8364	-28.6490	29.8219	30.2431	30.4518
BINPI *stump*	-29.7880	-30.0284	-29.0653	29.4203	30.0786	29.3984

Table 3.7: Simulation 7: main results

3.7 A real dataset: Boston Housing

Boston housing data set is a well known set of data by UCI machine learning repository (http://mlearn.ics.uci.edu/). It consists in 13 numerical variables and one dummy variable about housing values in suburbs of Boston and 506 instances. Data were deleted from three variables (median value of owner-occupied homes, proportion of residential land zoned for lots over 25,000 sq.ft, percentage of lower status of the population) according to the conditional probability as in equation 3.4. Following table summarizes main results of the imputation process in terms of mean, standard deviation and root mean squared error (the latter between fitted values and the previously deleted values from the original dataset).

	μ_1	μ_2	μ_3	σ_1	σ_2	σ_3	RMSE
True	11.3636	12.6531	22.5328	23.3225	7.1411	9.1971	
UMI	11.1365	12.8810	23.6695	18.8604	5.1030	8.0535	0.4680
PI	11.1015	15.6522	15.5894	11.9853	4.4478	3.7857	0.9753
INPI	11.2378	11.9624	22.0010	19.4274	5.1816	8.9793	0.2653
BINPI	11.3368	12.6649	22.4168	20.0260	6.5952	9.1241	0.1128

Table 3.8: Boston Housing dataset: main results

As it can be seen, always $BINPI$ algorithm works better than other techniques in terms of mean, standard deviation and root mean squared error. In general, non parametric techniques tend to impute missing values better than the parametric ones. The use of AdaBoost algorithm allows a more accurate imputation. Moreover Fast algorithm is preferred, specially when it is used as weak learner, in terms of computation cost of the overall procedure.

3.8 Concluding remarks

The incremental non parametric imputation provides a more accurate imputation compared to other standard methods. The main results of both the simulation study and the real dataset can be outlined as follows:

- The non parametric incremental imputation method seems to impute missing values better than other classic techniques;

- Within the framework of the incremental imputation approach, the imputation of a variable at turn is preferred to the imputation of a single data at turn; this can be understood by thinking that in the incremental imputation approach, filling in in the data matrix more informations is better than only one at turn;

- The imputations via boosting algorithm are more accurate than the ones obtained from single tree-based models. As it is well known [13], ensemble methods destroy the interpretation of a tree-based structure because they aggregate several different trees, but if what we want is a robust and accurate prediction, they work really well.

In the future, error back-propagation in the framework of the incremental imputation needs to be investigate. In other words, an error back-propagation, that can occur in the framework of ensemble methods, and that is part of the incremental approach where the first imputation would be characterized by a statistically elevated degree of error, is likely because the information of every variable in the incremental process results conditioned to the degree of completeness of themselves.

Chapter 4

Robust Incremental Tree-based Imputation for Data Fusion

4.1 Introduction

Data Fusion and Data Grafting are concerned with combining files and information coming from different sources [97]. The problem is not to extract data from a single database, but to merge information collected from different sample surveys. The term fusion is used in this extent. The typical data fusion situation formed of two data samples, the former made up of a complete data matrix X relative to a first survey, and the latter Y which contains a certain number of missing variables. The aim is completing the matrix Y beginning from the knowledge acquired from the X. As a consequence, the Data Fusion can be considered as a particular case of data imputation framework, with the difference that in this case a group of instances is missing as they have not been observed. In literature, several approaches have been introduced dealing with Data Fusion:

- the *classical approach* such as regression models, general linear models and logistic regression [72];
- the *implicit approach* based on the concept of similarity among the observations deriving from different sources [2];
- the recent *non parametric approach* that uses non standard regression techniques to impute missing values [7].

In this section, an innovative methodology for Data Fusion, based on a incremental imputation algorithm using tree-based models, is proposed. The goal of this approach is the definition of the correlation structure which joins the two data matrices that are object of the fusion. A recursive use of robust segmentation trees validated by boosting iterations [47] will be considered. This way, a tree-based incremental data fusion algorithm is defined which proceeds, step by step, to the completion of the missing instances. This methodology allows to overcome the situation characterized by the presence of heterogeneous kinds of unobserved variables. As a matter of fact, the tree-based models give the possibility to work, at the same time, with both qualitative and quantitative data.

4.2 Data Fusion framework

Data Fusion problem can be formalized in terms of two data files [2]. The first data file consists of a whole set of $p + q$ variables measured on n_0 individuals. This data file is called *donor file.* The second data file, usually named *receptor file*, consists in a subset of p variables measured on n_1 units (fig. 4.1).

So, the problem is to merge two different and independent databases. We could imagine two independent surveys named survey A

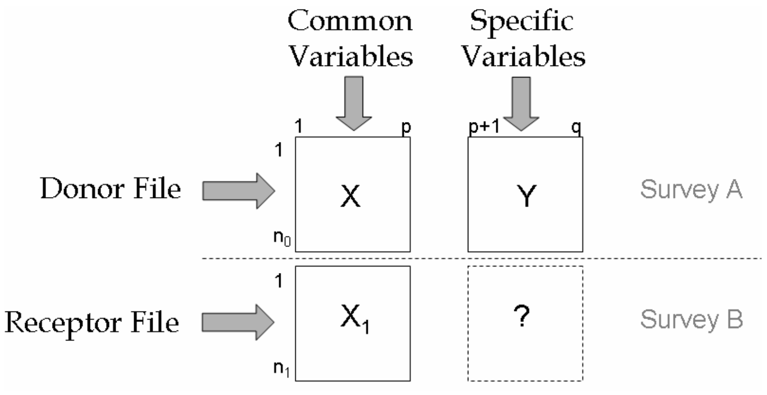


Figure 4.1: Data Fusion mechanism

and survey B. Say that survey A has been collected in a given supermarket, and say that survey B has been collected in a different supermarket of the same chain. The first set of variables X and X_1 are common to both supermarkets, whereas the other set Y is specific of Survey A. How would customer from second supermarket answers if we asked the same questions?
The interest in Data Fusion becomes clear at ones: Data Fusion could be used when there is lack of informations, to cut down costs, to take profit of existing data, and so on.

Before proceeding with the fusion, pre-fusion conditions should be verified, indeed the internal relationships of common variables X and X_1 should show a stable pattern [12].
Rius *et al.* [93, 94] suggests to verify initial conditions by identifying the common group of variables which define a similar representation subspace for both data files. Authors choose the common variable space from the different data sets by performing a Principal Component Analysis on X and then they use a branch-and-bound procedure

to eliminate variables in order to find a minimal set of variables of the common group. In a following step, they analyze the stability of the common space by bootstrap replications to assure that the association of the common variables is the same.

4.3 Models for data fusion

In the framework of Data Fusion, usually on distinguish between explicit models and implicit models [97].

- **Explicit models** are known as models based on variables. With explicit models, a model is used to connect Y variables with the X variables in the donor file and then applying this model in the receptor file.
- **Implicit models** are models based on individuals because for each individual belonging to the receptor file the k-nearest neighbours units of donor matrix are identified to transfer in some way the values of the Y variables to the receptor observation.

4.3.1 Explicit models

A quite natural idea is to use the cases with complete values of variables (X, Y) (namely, the donor file) to fit (linear) **regressions** of Y on X, and then use these regressions to impute the missing values by $\widehat{Y}$. When using regression in this manner some assumptions have to be verified:

- A good fit of the regression models;
- The constancy of relationship among predictors $\mathbf{X}$ and responses in both surveys;

- The null partial correlation of $\mathbf{Y}$ given $\mathbf{X}$.

A common problem in the general framework of missing data imputation when regression models are used to impute missingnes is the lack of variability of the genuine values [72], because one replaces unknown values about the regression hyperplane by imputed values on the hyperplane [7].

In order to use the **EM algorithm**, one has to specify a likelihood which requires information or an hypothesis about the generating mechanism of the data. In fact, EM algorithm provides an iterative way to maximize the likelihood function of incomplete data. Data imputed via EM algorithm suffer from the same lack of variability than the regression imputed values.

Multiple imputation techniques, based on Bayesian framework [95], allow to simulate the posterior distribution of the missing values by imputing each data with several values according to one or more estimation models. With multiple imputation techniques correct variances can be achieved, but these tools are really complex and time consuming [97].

Data Fusion (and missing data imputation) by **classification and regression trees** has well known advantages: trees give a unified treatment of continuous and categorical variables, they provide an easy tool for multiple imputation, they are relatively insensible to outliers. Barcena and Tusell [7] defined a data fusion procedure working with a multiple imputation via classification and regression trees named *forest climbing* algorithm.
Let $y_1, \ldots, y_j, \ldots, y_q$ be the specific variables belonging to Y block as shown in fig. (4.1) representing q response variables, and let $x_1, \ldots, x_p$ be the predictors.

The forest climbing algorithm works by building q classification (or regression) trees and then by dropping down the trees the individuals belonging to receptor file. The simplest multiple imputation rule consists in observing terminal nodes in which the i^{th} individual falls for each tree built according the number of response variable. Figure 4.2 shows this imputation process.
Suppose that in the fusion process we only have two specific variables, and suppose that the i^{th} individual falls down in the terminal node number 2 in the tree generated by the first response variable indicated by $T^{y_1}_{tn2}$ (the tree on the left side of the figure), and that the same individual falls down in the terminal node number 6 in the tree generated by the second response variable indicated by $T^{y_2}_{tn6}$ (the tree on the right side of the figure). The intersection $C^{i}_{2,6} = T^{y_1}_{tn2} \cap T^{y_2}_{tn6}$ of those two terminal nodes for the same i^{th} individual determines the multiple imputation rule, since the authors propose to impute for the i^{th} individual the Y values observed for cases in the training sample that also fall in $C_{2,6}$.
If no case in the training sample belongs to one particular intersection, the algorithm "climbs" the trees replacing one node at time by its father until one intersection is verified.

4.3.2 Implicit models

One example of implicit model for Data Fusion is the **nearest neighbour imputation**. In this way the idea is to replace the missing values in one case by those of another case in some sense close to it, according to a pre-defined notion of closeness in the space of common variables. Nearest neighbours imputation avoids incoherent estimations since the copied values belong to real observations [97] but, in order to define the closeness which could not be straightforward for categorical variables, the definition of a suitable distance measure is needed .

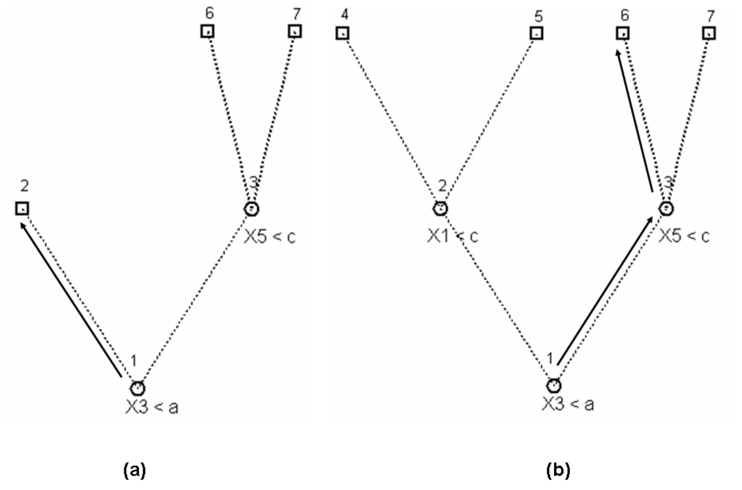


Figure 4.2: Multiple imputation by trees.
The arrows represent the path for the same i^{th} individuals for two response variables. (a) The path for the tree built with the first response variable; (b) The path for the tree built with the second response variable

The use of **factorial techniques** as Multiple Correspondence Analysis or Principal Component Analysis provides a Data Fusion procedure through a *file grafting* process [93, 3, 4]. File grafting consists in putting into the same factorial space the information coming from the (common variables of the) two independent data sets. Provided that the pattern of relations of X and X_1 (see fig. 4.1) is stable, it is possible to define a common sub-space for both data files by performing a PCA (or a MCA) on X and then by positioning the elements of X_1 upon the same reference space previously obtained. The n_1 individuals belonging to the X_1 matrix are positioned as supplementary points,

and the distance between these individuals should be a good approximation of the true distance among these supplementary points.
Once that this grafting procedure is done, Data Fusion consists on applying a k-nearest neighbors algorithm between the n_1 individuals belonging to the receptor file (which are projected as supplementary points on the common sub-space created by the factorial technique) and the n units belonging to the donor file.

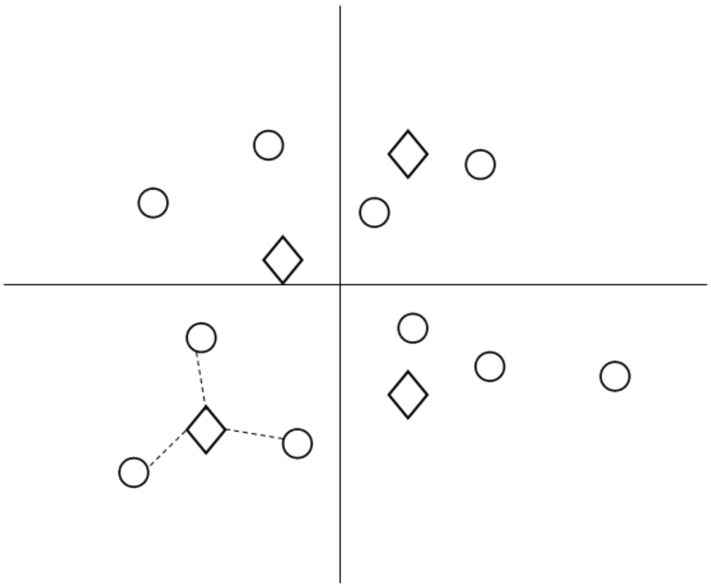


Figure 4.3: Neighbourhood in the factorial space.
Circles are individuals belonging to X matrix whereas rhombus are supplementary points, namely individuals belonging to receptor file. For each individual of X_1, neighbourod is defined by its k-nn in the cloud of individuals of X.

Finally, imputed values for the fusion process are the mean values of Y variables in the neighbourhood.

If a dependence relationship between specific and common variables is assumed, in the sense that Y block depends on X block, a Constrained Principal Component Analysis [27] instead of a PCA can be used as preliminary grafting step for Data Fusion. In this case, named Data Fusion through Non Symmetrical Grafting [90], the neighbourhood necessary for the fusion is computed onto the "asymmetric" reference sub-space generated by dependence structure of Y on X.

4.4 Robust Incremental Imputation algorithm for Data Fusion

The incremental approach, showed in the previous chapter, is integrated in a recursive methodology for data fusion that is called **Robust Tree-based Incremental Imputation** (RTII).
Figure 4.4 describes the main steps of the proposed imputation algorithm for Data Fusion.
Let the donor file be formed by **A** and **B** blocks, let the receptor file be made by the **C** block, and let **D** be the block to impute.
Common variables $x_1, ..., x_p$ are represented by **A** and **C** blocks, whereas **B** block symbolizes specific variables $y_1, ..., y_q$.

- **Step 0**.
  - Sort **B** block according to the dependence link with block **A**. Build a supervised tree for each y columns, then sort columns according to the previously obtained best tree.
- **Step 1**.

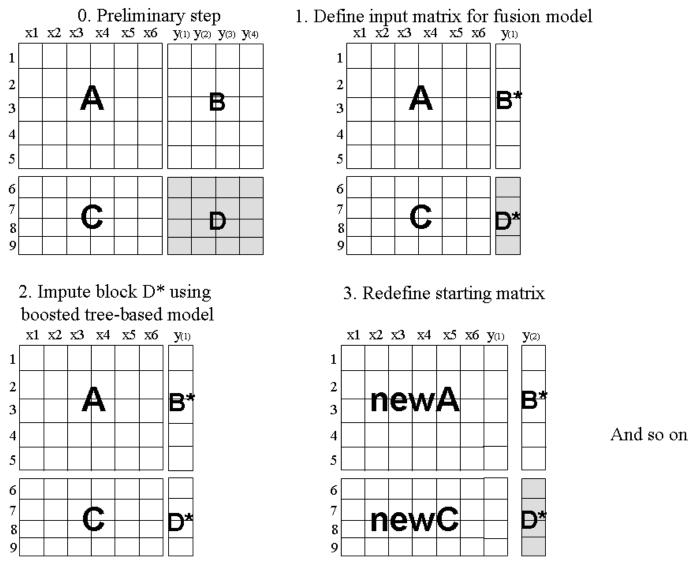

0. Preliminary step
1. Define input matrix for fusion model
2. Impute block D* using boosted tree-based model
3. Redefine starting matrix
And so on
A
B
C
D
B*
D*
newA
newC

Figure 4.4: RTII algorithm. (Grey blocks are missing values)

- Define the input matrix for fusion model; the first ordered y variable belonging to **B** block is the response and the complete variables (**A** block) are the predictors;

- **Step 2**.
  For $k = 1 : q$
  - Build a supervised tree with $V - fold$ AdaBoost iterations using blocks **A** as predictor and $\mathbf{B}^k$ as response variable, and use **C** to impute missing block $\mathbf{D}^k$;
  - Add $\mathbf{B}^k$ block to **A** to make the **newA** block and add the imputed block $\mathbf{D}^k$ to **C** one to form the **newB** block;

- **Step 3**.

  - The matrix is re-defined to impute a new column.
  - back to step 1 until all missing variables are completed;

- **Output**.
  - All variable belonging to **D** block are imputed

Preliminary step prepares the data matrix to the Fusion process. Specific variables are sorted according to their dependence link with common variables. A regression or classification tree is built for each specific variable, so these variables are sorted according to the best obtained root mean squared error (or the lower misclassification ratio for categorical case). The first sorted variable belonging to the block of the specific ones play the role of response variable in the first iteration of the iterative imputation process. A supervised tree -as weak learner for V-fold AdaBoost iterations- is built, as a consequence the common variables belonging to the receptor file are used to impute the first missing variable. Both this variable and the specific one are included in the complete block and then the second sorted variable belonging to the specific ones is used as response variable. The iterative process ends when all variables are imputed.

4.5 Simulation study: numerical imputations

The performance of the proposed method based on *Robust Tree-based Incremental Imputation* (RTII) algorithm has been evaluated in a simulation study.
First example regards the imputation of numerical values, the second the case of a mixture of variables (both numerical and categorical). In particular in the latter case, classical procedures cannot work, while

RTII algorithm executes all iterations without problems since tree-based models deal with both numerical and categorical instances. The goal is to estimate punctual values of cases to be imputed, and then to check the overall imputation procedure comparing the imputed variables with the control set in terms of their distribution. So, goodness imputation measures are the mean and standard deviation of imputed variables, the *root mean squared error* of used method, a measure of equality of mean between imputed and control variables using t-test, a measure of equality of variances between imputed and control variables using Fisher's test about equality of variances.
Performance of RTII algorithm has been compared with other methodologies such as Parametric Imputation via Multiple Regression (PI), Non Parametric Imputation by Standard Tree (Tree), PCA Fusion according to the Aluja et al.'s approach in [3, 4].
Table 4.1 shows the numerical simulation settings.
Simulation study has been defined thinking to reliable situations in which Data Fusion can be functional, i.e. when the donor file is a set of socio-economics variables (i.e., age, gender, income, job, etc.). For that reason, a simulated dataset was built using different random distributions for the set of common variables (Discrete uniform, Normal, Continue uniform), whereas specific variables were generated in both cases without relationship with common variables (according to normal and uniform distribution) and with linear link with other variables. Entire data sets were randomly splitted in two sub-sets (donor file and receptor file), then the part of specific variables belonging to receptor file was deleted from the data set and used as control set to check the goodness of imputation.

Simulation 1	
Donor file: 500 observations; Receptor file: 300 obsvervations; Missing values: 1200	
Common variables	*Specific variables*
X_1 uniform in {18,65}	$Y_1 = k + 0.3X_4 - 0.8X_1$
X_2 uniform in {1,4}	$Y_2 = k - 0.5X_5 + 0.1X_4 + 2X_2$
X_3 uniform in {1,3}	$Y_3 \sim N((X_1 - X_2), exp(0.7X_3 + 0.3X_1)$
$X_4 \sim N(100, 10)$	Y_4 uniform in [0,100]
$X_5 \sim N(500, 50)$	
Simulation 2	
Donor file: 400 observations; Receptor file: 200 observations; Missing values: 1000	
Common variables	*Specific variables*
X_1 uniform in {18,65}	$Y_1 = k + 0.8X_2 - 0.2X_4$
X_2 uniform in [10,100]	$Y_2 = k + 0.2X_3 + 0.3X_1 + 3X_6$
X_3 uniform in [0,100]	$Y_3 = k + 0.6X_5 + 0.5X_2$
$X_4 \sim N(200, 30)$	Y_4 uniform in [50,250]
$X_5 \sim N(700, 80)$	$Y_5 \sim N(100, 10)$
X_6 *dummy* variable	

Table 4.1: Simulation settings for numerical case

Figures 4.5 and 4.6 show test error progress through AdaBoost iterations. The error of boosted tree is always lower than error of single tree, and it seems to stabilize at the last tens of iterations, except for the first variable of simulation 2 which reaches a stable rate of error after about 60 iterations.

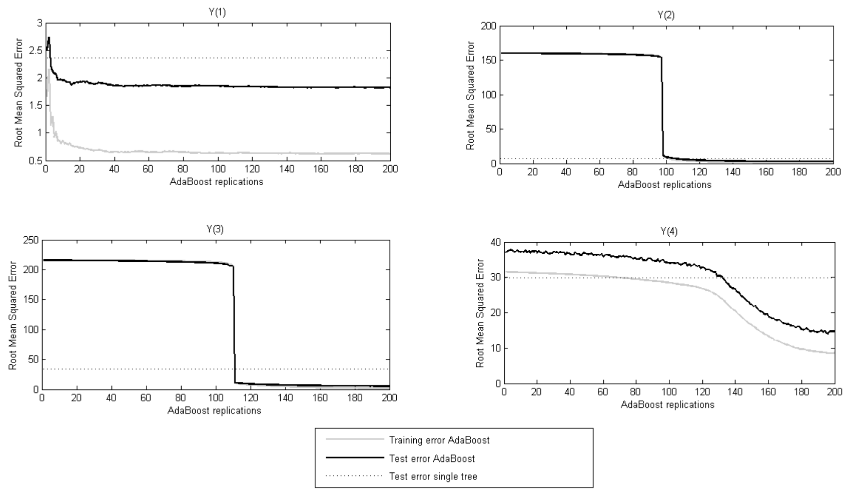


Figure 4.5: Simulation 1: Test error progress through AdaBoost iterations

Tables 4.2 and 4.3 show the results of RTII algorithm in comparison with some standard data fusion technique. For each simulated data set, tables show in the first two rows the results in terms of mean and standard deviation of the imputed variables. All mean values are close to the true means for all techniques, as well as some difference can be seen in terms of difference in standard deviation. Third row shows the root mean squared error. Always RMSE of RTII algorithm is lower than the one of the other methods. Imputations are evaluated also in terms of t-test for equality of means and F-test for equality of variances between imputed and control variables. Always t-test is not significant, as well as F-test is not significant for tree-based methods (”boosted” and not) in comparison with other techniques (see specially table 4.3).
As well-known a common problem concerning imputation methods is to reduce, in a significant way, the variance of imputed distribution.

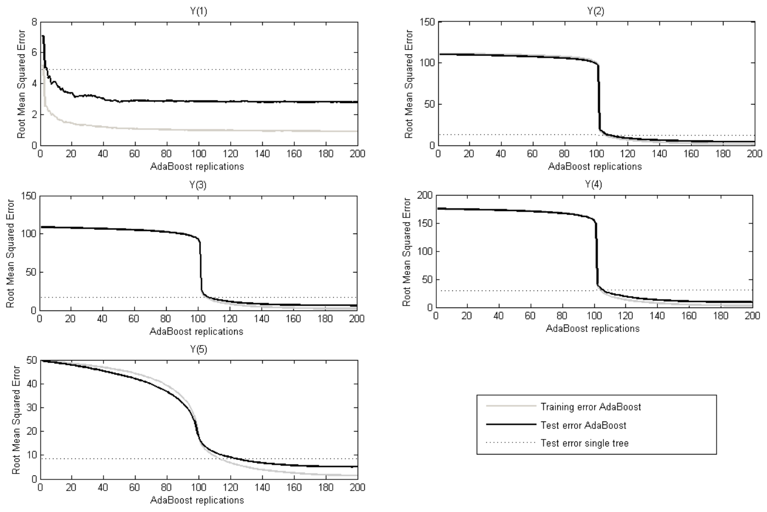


Figure 4.6: Simulation 2: Test error progress through AdaBoost iterations

Considering this aspect, a method gives good results when the imputation process does not involve a significant reduction of variance in comparison with missing distribution [72]. According to this point of view, Fisher's test about equality of variances has been used to compare the goodness of the different considered methodologies.
Regarding the tables, it can be noticed how all variables imputed via Multiple Regression and Factorial approach very often have a P-value of Fisher's test close to zero, which drives to refuse the null hypothesis about equality of variances. On the contrary for the imputation via RTII algorithm and standard tree, the null hypothesis is never refused (table 4.3) or the ratio forming the statistical index is sensitively lower (table 4.2).
In our opinion, this is an important remark because Tree-based methods, as non parametric tools, are determinant not only for the imputation of missing values, but above all in terms of variability reconstruc-

tion. Boosting algorithms make the estimate of missing values more robust, as it can be seen looking at lower root mean squared errors.

			Y_1	Y_2	Y_3	Y_4
RTII	Mean		96.003	253.980	39.688	51.837
	Standard deviation		10.671	24.390	3.098	23.601
	Root Mean Squared Error		1.794	2.872	3.927	13.746
	Fisher's Variance Test	F stat	1.047	2.872	3.927	13.746
		P-value	0.347	0.371	0.000	0.000
	Compare Means Test	T stat	-0.141	-0.069	0.041	0.175
		P-value	0.444	0.473	0.483	0.430
Classical Tree	Mean		95.852	253.990	39.801	49.447
	Standard deviation		10.385	22.650	0.000	0.000
	Root Mean Squared Error		2.801	4.660	4.934	29.884
	Fisher's Variance Test	F stat	1.109	1.075	2.19E+28	3.33E+28
		P-value	0.187	0.267	0.000	0.000
	Compare Means Test	T stat	0.156	0.373	1.146	-1.278
		P-value	0.438	0.355	0.126	0.101
PCA (Aluja et al approach)	Mean		96.569	255.000	39.843	50.176
	Standard deviation		7.220	4.967	0.379	4.384
	Root Mean Squared Error		7.395	23.472	4.919	30.998
	Fisher's Variance Test	F stat	2.281	23.071	169.870	46.501
		P-value	0.000	0.000	0.000	0.000
	Compare Means Test	T stat	0.579	0.791	0.834	-0.507
		P-value	0.281	0.215	0.202	0.306
Multiple Regression	Mean		95.007	254.080	38.734	50.210
	Standard deviation		10.817	23.858	0.312	5.419
	Root Mean Squared Error		5.971	6.280	7.880	30.940
	Fisher's Variance Test	F stat	1.016	1.000	251.190	30.431
		P-value	0.445	0.500	0.000	0.000
	Compare Means Test	T stat	-0.140	0.102	0.451	-0.371
		P-value	0.444	0.459	0.326	0.355
True mean			96.132	253.890	39.605	51.060
True standard Deviation			10.904	23.858	4.942	29.893

Table 4.2: Simulation 1: main results

			Y_1	Y_2	Y_3	Y_4	Y_5
RTII	Mean		105.550	206.470	313.250	145.700	100.580
	Standard deviation		21.866	54.624	69.789	52.529	39.739
	Root Mean Squared Error		3.201	4.192	6.309	10.344	8.439
	Fisher's Variance	F stat	1.083	1.006	1.005	1.161	1.117
	Test	P-value	0.287	0.482	0.486	0.146	0.218
	Compare Means	t stat	0.132	0.027	-0.084	-0.029	0.043
	Test	P-value	0.447	0.489	0.467	0.488	0.483
Classical Tree	Mean		104.200	205.280	314.550	148.190	101.040
	Standard deviation		22034	54.503	69.611	50.490	39.247
	Root Mean Squared Error		6.176	8.097	11.701	18.659	15.035
	Fisher's Variance	F stat	1,166	1.023	1.013	1.116	1.097
	Test	P-value	0.140	0.436	0.463	0.220	0.258
	Compare Means	t stat	-0.446	-0.002	-0.073	0.385	0.375
	Test	P-value	0,328	0.499	0.471	0.350	0.354
PCA (Aluja et al approach)	Mean		104.170	205.190	311.380	145.890	99.986
	Standard deviation		5.454	26.092	35.205	10.522	14.422
	Root Mean Squared Error		21.762	46.841	57.996	54.48	38.875
	Fisher's Variance	F stat	17.840	4.462	3.993	28.763	8.441
	Test	P-value	0.000	0.000	0.000	0.000	0.000
	Compare Means	t stat	-0.673	-0.309	-0.392	0.027	-0.009
	Test	P-value	0.251	0.379	0.347	0.489	0.497
Multiple regression	Mean		103.030	206.753	315.550	146.800	102.210
	Standard deviation		28.006	68.402	81.186	68.051	52.131
	Root Mean Squared Error		17.743	19.979	15.960	24.002	29.218
	Fisher's Variance	F stat	1.478	1.540	1.536	1.541	1.548
	Test	P-value	0.003	0.001	0.001	0.001	0.001
	Compare Means	t stat	-0.118	0.001	-0.002	0.004	0.047
	Test	P-value	0.453	0.500	0.499	0.498	0.481
True Mean			105.300	206.520	313.570	145.780	100.010
True Standard Deviation			23.035	55.114	70.349	56.432	41.901

Table 4.3: Simulation 2: main results

4.6 Simulation study: mixed variables imputation

A more realistic case occurs when datasets contain both numerical and categorical data. For this reason a dataset in this way characterized was simulated. As in previous section, several random distributions were used to simulate common variables, as well as different relationships bind together specific and common variables.
Table 4.4 shows the simulation setting for the mixed variable imputation. Both common and specific variables are multiclass categorical (both ordered and unordered), binary and numerical.

In this case, Factorial techniques cannot be suitably used, as well as classical regression models fail the goal of a unique multiple imputation of missing data.

Simulation 3	
Donor file: 600 observations; Receptor file: 200 observations; Missing values: 600	
$G \sim N(1000, 300)$:	$W{=}k + 1.3X_5 - 0.32X_4 + exp(0.5X_2 - 0.7X_3)^{-1}$:
$X_1 = 1$ if $G < 10^{th}$ percentile;	$Y_1 = 1$ if $W < 15^{th}$ percentile
$X_1 = 2$ if $10^{th} \leq G < 25^{th}$ percentile;	$Y_1 = 2$ if $W \geq 53^{th}$ percentile;
$X_1 = 3$ if $25^{th} \leq G < 75^{th}$ percentile;	$Y_1 = 3$ if $15^{th} \leq W < 35^{th}$ percentile;
$X_1 = 4$ if $G \geq 75^{th}$ percentile	$Y_1 = 4$ if $35^{th} \leq W < 53^{th}$ percentile;
$X_2 \sim N(1500, 450)$	$Y_2 = 1$ if $X_5 \cap X_4 + exp(0.3X_2)^{-1} <$ median value
X_3 uniform in [1, 350]	$Y_2 = 2$ if $X_5 \cap X_4 + exp(0.3X_2)^{-1} \geq$ median value
X_4 uniform in 1, 10	$Y_3 = k$+uniform in [10, 350]+$exp(0.2X_3 - 0.5X_5)^{-1} - 1.5X_6$
$X_5 \sim Bin(n,$ uniform in [0,1])	
X_6 *dummy* variable	

Table 4.4: Simulation setting for mixed variables imputation case

Figure 4.7 shows the test error progress through AdaBoost iterations for each simulated specific variable.
Goodness of imputation measure for categorical variables is the misclassification ratio whereas, as in the previous section, it is the root mean squared error for the numerical case.
Note how training error for categorical cases reaches to zero after less than 20 iterations. Note also how test error of boosted classifiers seems to be stable after about 100 iterations for all imputation cases.

Tables 4.5 and 4.6 show the classification table respectively for the multiclass and the binary imputation problems.
The misclassification ratio for the multiclass variable is equal to 4%. Looking at table 4.5, column totals compared to row totals show a

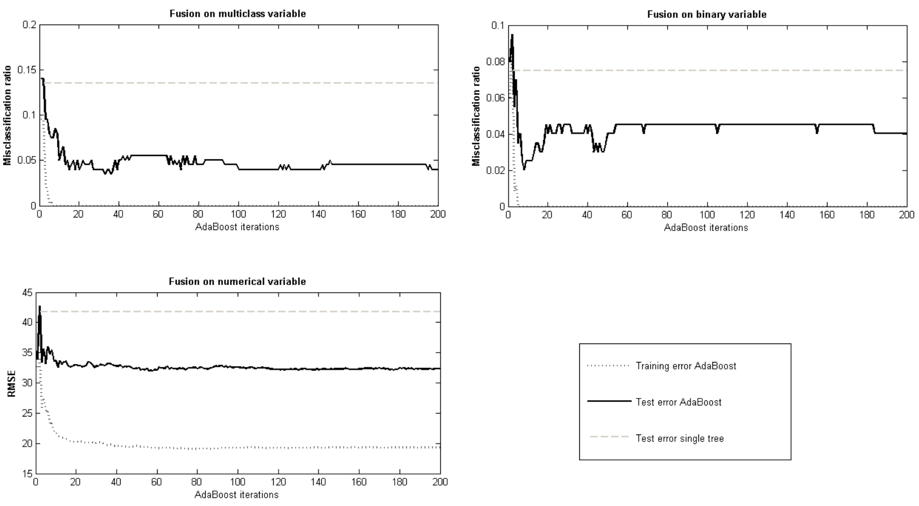


Figure 4.7: Simulation 3: test error progress through AdaBoost iterations

good approximation of the distribution of this variable.
The better predicted category is the second one with an error rate equal to 1.05%, whereas the worst predicted class is the first one with a misclassification ratio equal to 12%.

		Fitted values					
		1	2	3	4	Total	Error
Control values	1	22	0	0	3	25	0.1200
	2	0	94	1	0	95	0.0105
	3	0	0	28	3	31	0.0968
	4	0	0	1	48	49	0.0204
Total		22	94	30	54	200	0.0400

Table 4.5: Classification table for four-classes variable fusion imputation

Table 4.6 shows the classification table of the binary variable. Misclassification ratio is equal to 3.5% and the boosted classifier classifies the first category better than the second one.

		Fitted values			
		1	2	Total	Error
Control values	1	96	2	98	0.0204
	2	5	97	102	0.0490
Total		102	98	200	0.0350

Table 4.6: Classification table for binary variable fusion imputation

Figure 4.8 shows the box-plot for both control and imputed numerical variable. As it can be observed, both boxes are similar, even if the one concerning the imputed variable seems to suggest less variability with respect to the other one.

Table 4.7 shows the overall good performance of variable imputed by RTII algorithm in terms of both mean and standard deviation compared to these true indexes. Table shows also *t*-test and F-test about equality of respectively mean and variance between control and inputed variable. Fisher's test suggest a good variability reconstruction of the variable under imputation.

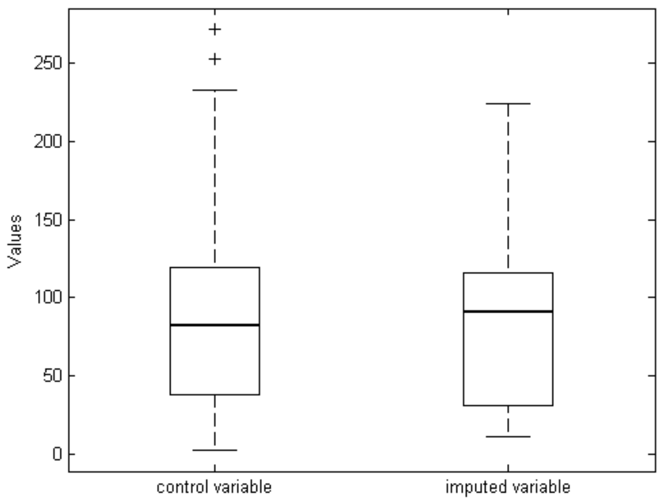


Figure 4.8: Box-plot for both control and imputed variable

Mean	88.0313
Standard deviation	58.9122
Root mean squared error	32.3287
True mean	86.7963
True standard deviation	57.5933
t-stat	0.2120
P-value	0.4161
F-stat	1.0463
P-value	0.3749

Table 4.7: Main results about numerical variable

4.7 Concluding remarks

Data Fusion can be considered as a special case of data imputation where the values to be imputed are those allowing the merging between two different sample surveys. This section has provided a methodology for Data Fusion characterized by three features;

- it considers an explicit non-parametric tool using a tree-based model;
- the recursive partitioning for data imputation leads to the use of an incremental approach where more and more information is added to the data matrix (see section 3);
- the tree validation is robust since boosting iterations are performed.

The overall method, called *Robust Tree-based Incremental Imputation*, presents two special advantages;

1. it can be considered for a mixed data structure that includes both numerical and categorical variables;
2. it allows to better reconstruct the imputed variable distribution in terms of both mean and variance.

As we remarked at the end of section 3, a possible error back propagation due to the boosted incremental imputation philosophy needs to be investigated.
Moreover, Data Fusion is more ambitious than "simple" missing data imputation: we want to impute never collected values! For this reason pre-fusion conditions need to be more and more deepened. These aspects will be studied and developed in the next future, always by recalling Saporta's words in terms of Data Fusion: *one has to be very*

careful when using data which are estimates and not observations: they should never used at an individual level. A perverse consequence of data fusion techniques may result in less effort to collect data, since we may invert them scientifically [97].

Chapter 5

Distance-Based Multivariate Trees for Rankings

5.1 Introduction

The main achievement of this chapter is the use of a multivariate tree-structure to understand which predictors and which interactions of predictors are the most significant to explain the response variable when this is constituted by rank order preference data or paired comparison rankings. A multivariate tree-based structure is the natural extension of univariate classification and regression trees when the response variable is multivariate. Several approaches to this framework have been developed in the last few years [105, 29, 70, 63] considering as response variable multivariate distributions. Conceptually, a ranking cannot be considered as a multivariate distribution but, better, as an unique multidimensional"entity".

5.2 Multivariate regression trees

A Multivariate Regression Tree is the natural extension of univariate regression trees. Let (Y, X) be a multivariate random variable where X is a set of K categorical or numerical predictors $(X_1, ..., X_k, ..., X_K)$ and Y is the set of J response variables $(Y_1, ...Y_j, ..., Y_J)$. Several splitting rules have been investigated to be used in MRT, and consequently several impurity measures, as well:

$$\sum_{i,j} (y_{ij} - \bar{y}_j)^2 \tag{5.1}$$

namely the multivariate sums of squared deviations about the mean, where $\bar{y}_j = \frac{1}{N_t} \sum\limits_{x_i \in t} y_{i,j}$

$$\sum_{k=1}^{K} \left(\sum_{j=1}^{J} \eta^2_{Y_j|X_k} w_j \right) \tag{5.2}$$

in which η^2 is the Pearson's squared correlation ratio, and w_j is the proportion of total sum of squares of Y_j over the total sum of squares of all response variables,
i.e. $w_j = \frac{TSS_{Y\,j}}{\sum\limits_{j=1}^{J} TSS_{Y\,j}}$.

$$\sum_{i,j} |y_{i,j} - \widehat{y}_j| \tag{5.3}$$

namely the multivariate sums of absolute deviations about the median, where $\widehat{y}_j$ is the median at the node t.

$$\frac{1}{N_t}\sum_{i>j,j} d_{ij}^2\,(y : x \in t) \tag{5.4}$$

namely the sum of squared distances at the node t.
Aim of this work is the use of distance measures as splitting criteria to discriminate between individuals when Y is a data matrix of rank orders.

5.3 Distance-Based Multivariate Trees for Rankings

Building a tree-based structure with rankings as response variable requires the definition of an impurity measure and an assignment. In this framework, the best way to work with rankings is to define a suitable distance, which is sufficiently discriminatory to be used as impurity measure. An important observation is that building a MRT using the sum of squared Euclidean distances or the multivariate sums of squared deviations about the mean leads to the same results. Specifically we have

$$\begin{aligned}
&\sum_i\sum_j d_{ij}^2(X) = \sum_i\sum_j\sum_a (x_{ia}-x_{ja})^2\\
&= \sum_i\sum_j\sum_a ((x_{ia}-\bar{x}_a)-(x_{ja}-\bar{x}_a))^2\\
&= \sum_i\sum_j\sum_a (x_{ia}-\bar{x}_a)^2 + \sum_i\sum_j\sum_a (x_{ja}-\bar{x}_a)^2\\
&= n\sum_i\sum_a (x_{ia}-\bar{x}_a)^2 + n\sum_j\sum_a (x_{ja}-\bar{x}_a)^2\\
&= 2n\sum_i\sum_a (x_{ia}-\bar{x}_a)^2
\end{aligned}$$

where double product term is zero because a simple difference of values around its averages. From previous results we have:

$$2n\sum_i\sum_a\left(x_{ia}-\bar{x}_a\right)^2=\sum_i\sum_j d_{ij}^2\left(X\right)$$
$$\sum_i\sum_a\left(x_{ia}-\bar{x}_a\right)^2=\frac{1}{2n}\sum_i\sum_j d_{ij}^2\left(X\right)=\frac{1}{n}\sum_{i>j}\sum_j d_{ij}^2\left(X\right)$$

In conclusion, there is no difference in growing a MRT choosing impurity as in equations 5.1 or 5.5 using as metric the Euclidean distance. In the framework of preference rankings, problems can arise when someone has to choose a measure of distance between rankings. A general distance measure should at least ensure that equal preference structures must have zero distance, and as the difference in these structures increases their distance has to increase. Euclidean distance does not seem the best one to work with rankings, nevertheless other constraints have to be added to limit the choice of a feasible measure to be used with such structured response matrix variable.

5.3.1 The Kemeny distance

Kemeny [67] introduced several constraints that a suitable distance measure for rankings should satisfy:

1. The distance measure should be a metric, so it must satisfy the following properties:
   - non negativity: $d(A, B) \geq 0$
   - symmetry: $d(A, B) = d(B, A)$
   - triangular inequality: $d(A, B) + d(B, C) \geq d(A, C)$
2. The measure of distance should not be affected by a relabeling of the set of objects to be ranked;
3. If two rankings are in complete agreement at the beginning and at the end of the list and differ only in the middle, than the distance does not change after deleting these two rankings;
4. The minimum positive distance is one.

Kemeny introduced a convenient representation for a ranking of n objects. By choosing $n = 3$, one can represent the ranking A by a square matrix (called *score matrix*) $A = \{a_{ij}\}$, where i, j=1,2,...,n. He used the following convention

$$a_{ij} = \begin{cases} 1 & \text{if object } i \text{ preferred to the object } j \\ -1 & \text{if object } j \text{ is preferred to th object } i \\ 0 & \text{if they are tied} \end{cases}$$

For example, the ranking $[b\ \{ac\}]$ is represented by

$$\begin{array}{c|c|c|c|} & a & b & c \\ \hline a & 0 & -1 & 0 \\ \hline b & 1 & 0 & 1 \\ \hline c & 0 & -1 & 0 \\ \hline \end{array}$$

The relationship $a_{ij} = 1$ expresses that i is preferred to j. Such a preference relation must be **asymmetric** and **transitive**. Hence,

- if $a_{ij} = 1$, then $a_{ji} = -1$.
- if $a_{ij} = 1$, and $a_{jk} = 1$, then $a_{ik} = 1$.

The relationship $a_{ij} = 0$ expresses that i and j are tied. Such an equivalence relation must be **reflexive**, **symmetric** and **transitive**. That is,

- $a_{ii} = 0$
- if $a_{ij} = 0$, then $a_{ji} = 0$.
- if $a_{ij} = 0$, and $a_{jk} = 0$, then $a_{ik} = 0$

The two relations must be **consistent**, for example,

- if $a_{ij} = 1$, and $a_{jk} = 0$, then $a_{ik} = 1$
- if $a_{ij} = 0$, and $a_{jk} = 1$, then $a_{ik} = 1$

In short, these conditions can be expressed more simply as follows:

1. $a_{ij} = 1$, or -1.
2. $a_{ij} = -a_{ji}$.
3. if $a_{ij} \geq 0$ and $a_{jk} \geq 0$, then $a_{ik} \geq 0$; and $a_{ik} = 0$ only if both the others are 0.

These conditions are uniquely satisfied computing a distance in this way:

- If there are two judges that have to rate two objects B and A, distance counts 0 if there is judgement uniformity between the raters, i.e. B is preferred over A for both judges;

	First choice	Second choice
Judge A	🍌	🍐
Judge B	🍌	🍐

Table 5.1: The distance between two judges is 0

- Distance counts 2 if disagreement occurs in the preference of the judges, i.e. for a rater B is preferred to A and for the other A is preferred to B;

	First choice	Second choice
Judge A	🍌	🍐
Judge B	🍐	🍌

Table 5.2: The distance between two judges is 2

- Distance counts 1 if one judge prefers an object over the other, where the other does not make a decision between objects, i.e. B is preferred to A for the first rater while B and A are considered tie for the other one.

Although Kemeny never called such distance with a name, it is known as the Kemeny distance [114, 123].

	First choice	Second choice
Judge A		
Judge B	and	

Table 5.3: The distance between two judges is 1

To compute the Kemeny distance one can proceed as follow:
Let Y a n by k data matrix containing the preference patterns of n individuals for k objects, construct a n by $k(k-1)$ matrix P in which each column represents one pair of objects.
After the computation of

$$p_{i[KL]} = sign(y_{iK} - y_{iL})$$

The Kemeny distance is

$$_{kem}d_{ij} = \sum_{[KL]=1}^{\frac{1}{2}k(k-1)} \left| p_{i[KL]} - p_{j[KL]} \right|$$

It easily can be seen that Kemeny distance is in fact a city-block distance, while taking the square root of $_{kem}d_{ij}$ this measure fits into Euclidean space [114].

As impurity measure we choose the sum of Kemeny distances at the node t, because it is the unique satisfying all the previously indicated axioms:

$$i(t) = \frac{1}{N_t} \sum_{i>j,j} {}_{kem}d_{ij} \left(y : x \in t\right) \tag{5.5}$$

5.3.2 Consensus Ranking

Given m rankings of n objects, what ranking best represents the consensus opinion? So, the consensus ranking should be that point which is in best agreement with the set of selected rankings. Defining a consensus ranking is necessary in the framework of tree-based models, because we have to assign *class ranking* in each node (terminal and internal) of the tree-structure. Kemeny [67] defined two types of consensus rankings, namely the *median ranking* and the *mean ranking.*

Let $A_1, ...A_M$ be a set of points not necessarily distinct.
The *median ranking* of the set is that point (or these points) B for which

$$\sum_{i=1}^{m} {}_{kem}d\left(A_i, B\right) = \min \tag{5.6}$$

The *mean ranking* of the set is that point (or these points) B for which

$$\sum_{i=1}^{m} {}_{kem}d\left(A_i, B\right)^2 = \min \tag{5.7}$$

Unfortunately, finding the consensus ranking is known to be a NP-hard problem [42]. In fact, the number of orderings of n objects is closely approximated by:

$$N_{(n)} \approx \frac{C^{n+1}\left(n!\right)}{2} \tag{5.8}$$

where $C = 1/ln2$
The following table shows how the number of orderings increases when the number of the objects to be ranked increases.

Objects	# of orderings
2	3
3	13
4	75
5	541
6	4683
7	47293
8	545835
9	7087261
10	102E+08

For that reason, several heuristic methods to approximate the consensus ranking have been proposed. For example, the mean rank numbers for a set of options can be approximated by summing the rankings for each option, dividing the sum by the number of judges and rearranging the options according to increasing mean ranks [123]. To approximate the median ranking one can invert adjacents options in the mean ranking and check if the total number of inversions from individuals rankings diminishes; if it does not, the search stops, otherwise the current pair of options is inverted and the new ranking is the candidate for the solution [44].

5.3.3 Prediction error

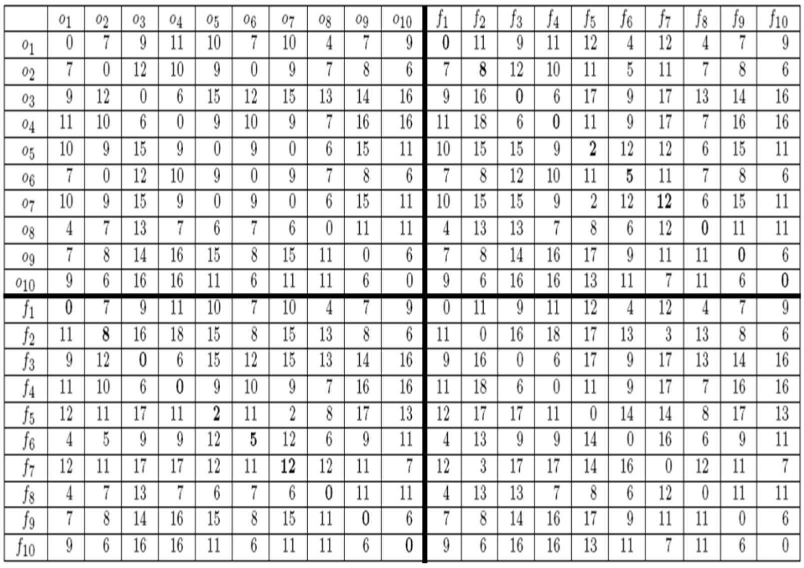
To evaluate the quality of a tree-based structure, the definition of a measure of error is necessary. Conceptually, a ranking cannot be considered as a multivariate distribution but, better, as an unique multidimensional "entity". For that reason, to evaluate the quality of a distance-based multivariate tree for rankings, a comparison of the distance between the rankings which contribute to build the tree-based structure and the fitted rankings needs to be made. Suppose that following table represents the rankings of ten people on six objects (in the table, the higher is the number the better is the position of the object in the order: i.e., in the first row the order is *b a d c e f*, in the second row the order is *e b a (cdf)*, and so on):

Objects					
a	b	c	d	e	f
4	5	2	3	1	0
2	4	0	0	5	0
4	2	2	5	2	0
4	3	0	5	0	2
2	3	1	2	1	5
5	4	2	2	0	0
3	4	0	1	5	2
4	5	1	3	0	2
2	4	2	1	2	0
4	5	1	0	3	1

Suppose again that the following table represents an imaginary fitted rankings of a tree-based classifier:

Objects					
fa	fb	fc	fd	fe	ff
4	5	2	3	1	0
4	4	1	1	1	1
4	2	2	5	2	0
4	3	0	5	0	2
2	4	1	2	1	3
4	4	1	1	1	1
2	4	1	2	1	3
4	5	1	3	0	2
2	4	2	1	2	0
4	5	1	0	3	1

Distance matrix D between observed and fitted rankings (computed using Kemeny distance) is the following:



	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
o_1	0	7	9	11	10	7	10	4	7	9	**0**	11	9	11	12	4	12	4	7	9
o_2	7	0	12	10	9	0	9	7	8	6	7	**8**	12	10	11	5	11	7	8	6
o_3	9	12	0	6	15	12	15	13	14	16	9	16	**0**	6	17	9	17	13	14	16
o_4	11	10	6	0	9	10	9	7	16	16	11	18	6	**0**	11	9	17	7	16	16
o_5	10	9	15	9	0	9	0	6	15	11	10	15	15	9	**2**	12	12	6	15	11
o_6	7	0	12	10	9	0	9	7	8	6	7	8	12	10	11	**5**	11	7	8	6
o_7	10	9	15	9	0	9	0	6	15	11	10	15	15	9	2	12	**12**	6	15	11
o_8	4	7	13	7	6	7	6	0	11	11	4	13	13	7	8	6	12	**0**	11	11
o_9	7	8	14	16	15	8	15	11	0	6	7	8	14	16	17	9	11	11	**0**	6
o_{10}	9	6	16	16	11	6	11	11	6	0	9	6	16	16	13	11	7	11	6	**0**
f_1	**0**	7	9	11	10	7	10	4	7	9	0	11	9	11	12	4	12	4	7	9
f_2	11	**8**	16	18	15	8	15	13	8	6	11	0	16	18	17	13	3	13	8	6
f_3	9	12	**0**	6	15	12	15	13	14	16	9	16	0	6	17	9	17	13	14	16
f_4	11	10	6	**0**	9	10	9	7	16	16	11	18	6	0	11	9	17	7	16	16
f_5	12	11	17	11	**2**	11	2	8	17	13	12	17	17	11	0	14	14	8	17	13
f_6	4	5	9	9	12	**5**	12	6	9	11	4	13	9	9	14	0	16	6	9	11
f_7	12	11	17	17	12	11	**12**	12	11	7	12	3	17	17	14	16	0	12	11	7
f_8	4	7	13	7	6	7	6	**0**	11	11	4	13	13	7	8	6	12	0	11	11
f_9	7	8	14	16	15	8	15	11	**0**	6	7	8	14	16	17	9	11	11	0	6
f_{10}	9	6	16	16	11	6	11	11	6	**0**	9	6	16	16	13	11	7	11	6	0

The error of this imaginary tree-structure can be computed by summing the Kemeny distances between each pair of observed rankings and fitted rankings, that is, summing over the diagonal elements of one of the two off-diagonal sub-matrices involving the distance between observed and fitted rankings. For example, as the ranking of the first person is perfectly fitted by the tree, the Kemeny distance between o_1 (observed 1) and f_1 (fitted 1) is 0; distance between o_2 and f_2 is 8, and so on. Calling this sub-matrix ${}_{off}D$, we define the prediction error at generic node t as

$$R(t) = \sum_{i=j} {}_{off}D_{ij}^{t} = tr\left({}_{off}D^{t}\right) \tag{5.9}$$

where ${}_{off}D^{t}$ is the distance sub-matrtix relative to the generic node t, and $R(t)$ is the error at t^{th} node. By the definition of impurity of the distance-based multivariate tree for rankings follows that the maximum error is the one computed in the root node, so a measure for the relative error in the generic t node is

$$R(t) = \frac{\sum\limits_{i=j} {}_{off}\mathbf{D}_{ij}^{t}}{\sum\limits_{i=j} {}_{off}\mathbf{D}_{ij}^{root}} = \frac{tr\left({}_{off}\mathbf{D}^{t}\right)}{tr\left({}_{off}\mathbf{D}^{root}\right)} \tag{5.10}$$

If $\tilde{T}$ is the set of all terminal nodes, the error of the overall tree-structure is

$$R(T) = \sum_{t \in \tilde{T}} R(t) \tag{5.11}$$

This definition of prediction error allows reaching for an error equal to zero if the tree perfectly fits, and equal to one if the tree is not discriminatory at all.

5.3.4 Searching for the consensus ranking

Emond & Mason [42] showed that an extension of Kendall's tau, named τ_X (tau extension) is equivalent to the Kemeny distance, while it is not true for the "classical" τ. The difference between Kendall's tau and this new correlation coefficient concerns only the way in which ties are handled. Briefly, for any ranking A of n objects Kendall defined a score matrix $\{a_{ij}\}$

$$a_{ij} = \begin{cases} 1 & \text{if object } i \text{ is ranked ahead of object } j \\ -1 & \text{if object } i \text{ is ranked behind object } j \\ 0 & \text{if objects are tied, or if } i = j \end{cases}$$

For example, if one person judges four objects (A,B,C,D) in this way

Ordering
B C
A
D

the score matrix is

	A	B	C	D
A	0	-1	-1	1
B	1	0	0	1
C	1	0	0	1
D	-1	-1	-1	0

Kendall's coefficient is then

$$\tau_b\left(A,B\right)=\frac{\sum_i\sum_j a_{ij}b_{ij}}{\sqrt{\sum_i\sum_j a_{ij}^2\sum_i\sum_j b_{ij}^2}} \tag{5.12}$$

Emond and Mason showed that the metric distance associated to this coefficient fails the triangular inequality. Representing the score matrix in the following way

$$a_{ij}=\begin{cases}1 & \text{if object } i \text{ is ranked ahead of object } j\text{, or if they are tied}\\ -1 & \text{if object } i \text{ is ranked behind object } j\\ 0 & \text{if } i=j\end{cases}$$

Score matrix relative to the previous ordering is

	A	B	C	D
A	0	-1	-1	1
B	1	0	1	1
C	1	1	0	1
D	-1	-1	-1	0

Emon and Mason define the new correlation coefficient as

$$\tau_X\left(A,B\right)=\frac{\sum_i\sum_j a_{ij}b_{ij}}{n(n-1)} \tag{5.13}$$

and they proved that it is equivalent to

$$1-\frac{2d(A,B)}{n(n-1)}$$

where $d(A,B)$ is the Kemeny distance.

Consensus ranking

Given m weak orderings of n objects, $A_1, ..., A_m$, where each ordering carries a positive weight w_k, consensus ranking S is the one that maximizes the weighted average correlation with the m input rankings, or equivalently is the one that minimizes the weighted average distance to the m input rankings.

$$maximize : \frac{\sum_{k=1}^{m} w_k \tau_X(S, A^k)}{\sum_{k=1}^{w_k}} \tag{5.14}$$

If $\{s_{ij}\}$ and $\left\{a_{ij}^k\right\}$ are the scoring matrices for S and A^k, the problem is:

$$maximize : \sum_{k=1}^{m} w_k \left\{\sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij} a_{ij}^k\right\} = maximize : \sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij} c_{ij} \tag{5.15}$$

Where $c_{ij} = \sum_{k=1}^{n} w_k a_{ij}^k$

Score matrix $\{c_{ij}\}$ is called by Emond and mason *Combined Input Matrix* (CI) because it is the result of a summation of each input ranking. So, as all the rankings information can be summarized in a single matrix, this way to find the best approximation to consensus ranking seems the more tractable. Moreover, score matrix $\{c_{ij}\}$ is a natural and accredited candidate to be the consensus ranking.
Authors conceived a branch-and-bound algorithm to maximize equation 5.15 by defining an upper limit on the value of that dot product. This limit, considering that the score matrix consists only of the values

1, 0 and -1, is given by the sum of absolute values of its elements:

$$V = \sum_{i=1}^{n} \sum_{j=1}^{n} |c_{ij}|$$

If a weak ordering of n objects is given as initial solution, it is possible compute the associated score matrix $\{s_{ij}\}$ and evaluate the value of expression 5.15. Then it is possible define an initial penalty P by subtracting this value from V. The problem is to search the set of all weak orderings of n objects to find those with the minimum penalty. This set can be divided into three mutually exclusive branches based on the relative position of the first two objects in the ordering represented in the initial solution, labeled as i and j. An incremental penalty for each of the branches can be calculated, by considering the corresponding elements c_{ij} and c_{ji} of the CI matrix, as follow;
Let δP be the incremental penalty:

- object i is preferred to object j (Branch 1):
  if $c_{ij} > 0$ *and* $c_{ji} < 0$, then $\delta P = 0$
  if $c_{ij} > 0$ *and* $c_{ji} > 0$, then $\delta P = c_{ji}$
  if $c_{ij} < 0$ *and* $c_{ji} > 0$, then $\delta P = c_{ji} - c_{ij}$

- object i is *ex aequo* with object j (Branch 2):
  if $c_{ij} > 0$ *and* $c_{ji} < 0$, then $\delta P - c_{ji}$
  if $c_{ij} > 0$ *and* $c_{ji} > 0$, then $\delta P = 0$
  if $c_{ij} < 0$ *and* $c_{ji} > 0$, then $\delta P = -c_{ij}$

- object j is preferred to object i (Branch 3):
  if $c_{ij} > 0$ *and* $c_{ji} < 0$, then $\delta P = c_{ij} - c_{ji}$
  if $c_{ij} > 0$ *and* $c_{ji} > 0$, then $\delta P = c_{ij}$
  if $c_{ij} < 0$ *and* $c_{ji} > 0$, then $\delta P = 0$

If the incremental penalty for any branch is greater than the initial penalty, then stop considering it because all orderings on the branch

will have a total penalty larger than the initial one.
If the incremental penalty of a branch is less than (or equal to) the initial penalty, consider the next object in the initial solution and create new branches by placing this object in all possible positions relative to the objects already there.
Continue in this way (by including all other objects) until all branches are exhausted or until a branch is followed to the end.
A weak ordering of n objects whose total penalty is less than or equal to the (current) minimum penalty have been found. The current penalty is then updated and continue in this way until the entire set has been searched [42].

5.4 Simulation case study

The codes to run the Distance-Based Multivariate Tree for Rankings were wrote in MatLab environment. To evaluate the goodness of our code, a set of rankings was simulated to check if codes were correct. The following table shows the simulation settings.

Without considering ties, the set of 24 rankings coming from 4 objects was extracted with probability indicated in the last column. To get this probability distribution, a number in $[1, 24]$ was extracted from a uniform distribution 150 times, then a random number from standard normal distribution was added to the proportion of each extracted number. The sample size was equal to 274 units and a vector indicating each rank (without noise) was used as predictor. The following figure shows the resulting Distance-Based Multivariate Tree for Rankings (DBMTR), as well as table 5.4 summarizes the output of the tree (up to the terminal nodes).

Ranking				Probability
A	**B**	**C**	**D**	
1	2	3	4	0.069
1	2	4	3	0.017
1	4	2	3	0.044
4	1	2	3	0.035
4	1	3	2	0.065
1	4	3	2	0.055
1	3	4	2	0.033
1	3	2	4	0.001
3	1	2	4	0.060
3	1	4	2	0.342
3	4	1	2	0.045
4	3	1	2	0.058
4	3	2	1	0.067
3	4	2	1	0.054
3	2	4	1	0.013
3	2	1	4	0.030
2	3	1	4	0.068
2	3	4	1	0.067
2	4	3	1	0.023
4	2	3	1	0.065
4	2	1	3	0.042
2	4	1	3	0.026
2	1	4	3	0.060
2	1	3	4	0.001

As expected, there are 24 terminal nodes (one for each ranking), and the impurity at terminal nodes (namely, the kemeny distance within each termninal node) is equal to zero.

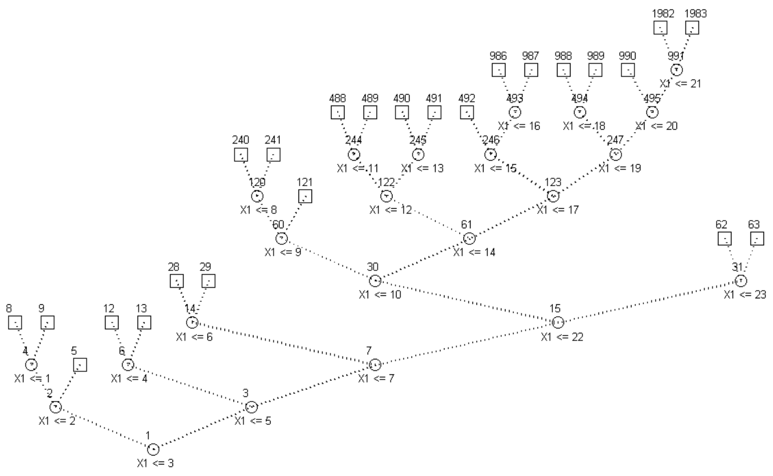


Figure 5.1: Example of DBMTR

Terminal nodes	# terminal node	Size at node	Ranking				Kemeny distance at node
			A	**B**	**C**	**D**	
1	5	5	1	4	2	3	0
2	9	3	1	2	4	3	0
3	13	19	4	1	3	2	0
4	29	15	1	3	4	2	0
5	121	13	3	1	4	2	0
6	241	7	3	1	2	4	0
7	489	19	4	3	1	2	0
8	491	9	3	4	2	1	0
9	987	25	2	3	1	4	0
10	989	11	2	4	3	1	0
11	1983	5	2	4	1	3	0
12	63	4	2	1	3	4	0
13	8	30	1	2	3	4	0
14	12	11	4	1	2	3	0
15	28	11	1	4	3	2	0
16	240	3	1	3	2	4	0
17	488	7	3	4	1	2	0
18	490	20	4	3	2	1	0
19	492	6	3	2	4	1	0
20	986	5	3	2	1	4	0
21	988	12	2	3	4	1	0
22	990	16	4	2	3	1	0
23	1982	5	4	2	1	3	0
24	62	13	2	1	4	3	0

Table 5.4: DBMTR output on simulated data

5.5 A real dataset: university rankings

University rankings dataset was analysed by Dittrich *et al.* to investigate paired comparison data concerning European universities and student's characteristics with the goal to show that university rankings are different for different groups of students [34].
A survey of 303 students studying at the Vienna University of Eco-

nomics was carried out to examine the student's preference of six universities, namely London, Paris, Milano, St. Gallen, Barcelona and Stockholm. The data set contains 23 variables. The first 15 digits in each row indicate the preferences of a student. For a given comparison, responses were coded by 1 if the first preference was preferred, by 2 if the second university was preferred, by 3 if universities are tied and by 4 if response was missing. All rows containing value 4 (missing response) were skipped, as suggested by authors. The first question was how to prepare the matrix of response variable $\mathbf{Y}$ to built the DBMTR. We have 6 universities and 15 variables indicating paired comparison. For each student, the number of time in which, in comparing i^{th} and j^{th} university the i^{th} university was preferred was counted.
List of the $\binom{6}{2} = 15$ paired comparison variables is:

- LP: comparison of London to Paris;
- LM: comparison of London to Milano
- PM: comparison of Paris to Milano
- LSg: comparison of London to St. Gallen
- PSg: comparison of Paris to St. Gallen
- MSg: comparison of Milano to St. Gallen
- LB: comparison of London to Barcelona
- PB: comparison of Paris to Barcelona
- MB: comparison of Milano to Barcelona
- SgB: comparison of St. Gallen to Barcelona

- LSt: comparison of London to Stockholm
- PSt: comparison of Paris to Stockholm
- MSt: comparison of Milano to Stockholm
- SgSt: comparison of St. Gallen to Stockholm
- BSt: comparison of Barcelona to Stockholm

In example, the following are 2 rows of the paired comparison matrix:

LP	LM	PM	LSg	PSg	MSg	LB	PB	MB	SgB	LSt	PSt	MSt	SgSt	BSt
1	3	2	1	2	1	1	2	1	1	1	2	1	1	2
1	1	2	1	1	1	1	2	2	2	1	3	1	3	1

In the first row, London is preferred to Paris, St. Gallen, Barcelona Stockholm (LP, LM, LSg, LB and LSt are always equal to 1), and there is no preference between London and Milano (they are tied); Milano is preferred to Paris (PM = 2), St. Gallen, Barcelona and Stockholm; and so on. So, response matrix $\mathbf{Y}$ for this example is:

L	P	M	Sg	B	St
4	0	4	1	2	1
5	1	3	0	4	0

The ranking of the first individual is [{*London Milano*} *Barcelona* {*St. Gallen Stochkolm*} *Paris*] as well as the ranking for the second unit is [*London Barcelona Milano Paris* {*St. Gallen Stochkolm*}].

The following 8 digits represent the subject-specific covariates, namely:

- X_1: (S) Main discipline of study: 1 = commerce; 2 = other;
- X_2: (Eng) Knowledge of English: 1 = good; 2 = poor;

- X_3: (Fra) Knowledge of French: 1 = good; 2 = poor;
- X_4: (Spa) Knowledge of Spanish: 1 = good; 2 = poor;
- X_5: (Ita) Knowledge of Italian: 1 = good; 2 = poor;
- X_6: (W) Full-time employment while studying: 1 = no; 2 = yes;
- X_7: (D) Intention to take an international degree: 1 = no; 2 = yes;
- X_8: (Sex) Sex: 1 = female; 2 = male;

Note that all predictors of this (real) data set are binary (yes/no, good/poor, etc.). Nevertheless the resulting tree-structure is quite extensive, as it can be seen looking at figure 5.5.
Circles represent parent nodes, as well as rectangles symbolize terminal nodes.
The number above nodes indicate node number, whereas strings below nodes indicate split variable.
The figure has to be read together with table 5.5.
The shortest path is the one getting to terminal nodes 10 and 11. Students having poor knowledge of French *and* good knowledge of Italian *and* willing to take an international degree express the ranking: [London Milano Paris Barcelona {St. Gallen Stochkolm}]. If they have no intention to take international degree, the preferred university is Milano, then London and St. Gallen are tied to the second place and all the other are tied at the third place.
In general, students with good knowledge of Italian and Spanish show preference for Milano (terminal nodes 11 and 29) and Barcelona (terminal node15) respectively. Paris is preferred by working-students that have good knowledge of French and Italian and that don't study business sciences, as well as it is preferred for students of commercial disciplines that speak Spanish. London is preferred by students

speaking good English and having intention to take an international degree. Second best are, in this case, Paris and sometimes Barcelona. Language skills govern the preferences for the universities. Probably Stockholm is less attractive than others because Swedish language is not well known as other languages.

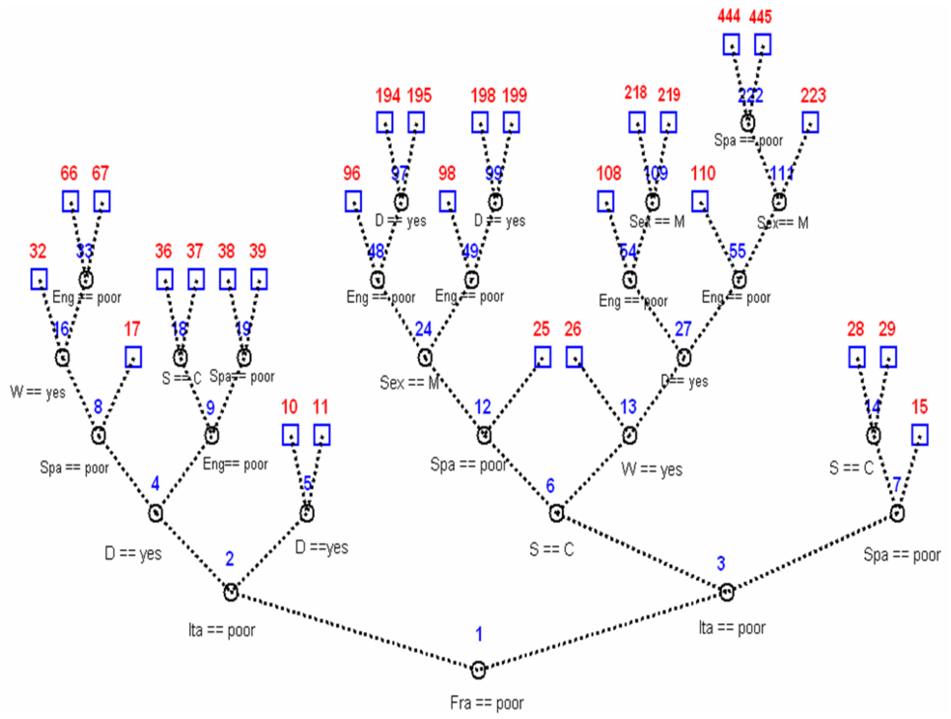


Figure 5.2: DBMTR on university rankings data

The last column of table 5.5 shows the impurity at each node, defined as the sum of within-node kemeny distance. Impurity at root node is

equal to 1336.90, whereas the overall impurity of the tree is equal to 51.38. The difference between these two measures is 1285.52, which means that the decrease in impurity is equal about to 96%.
The performance of the analysis in terms of prediction error as defined in equation 5.9 is quite poor: the error at the root node is equal to 14.98 whereas the error of the tree-structure is equal to 7.70 with the meaning that the error rate of the tree is equal to 51.42%.
Doubtless this depends on the nature of predictors (all binary variables), nevertheless according our opinion it is normal to obtain low error rates with this kind of response variable: recall that, considering ties, there are 4683 possible orderings (see equation 5.8) when 6 objects have to be ordered.

Node number	Size at node	Rule	Parent node	Ranking	Impurity at node
1	212	Knowledge of French: poor'	Root node	L P {M Sg B} St	1336.90
2	74	Knowledge of Italian: poor'	1	L B Sg {P M St}	168.30
4	59	Intention of international degree: yes'	2	L B Sg St {P M}	102.25
8	20	Knowledge of Spanish: poor'	4	L Sg {B St} {P M}	11.01
16	18	Full-time employment: yes'	8	L Sg B St {P M}	8.75
33	17	Knowledge of English: poor'	16	L Sg B St {P M}	7.93
9	39	Knowledge of English:poor'	4	L B P {M Sg St}	44.00
18	18	Main discipline of study: commerce'	9	L {B P} {M Sg St}	8.09
19	21	Knowledge of Spanish: poor'	9	{L B} Sg P {M St}	13.09
5	15	Intention of international degree: yes'	2	M L P {Sg B St}	5.97
3	138	Knowledge of Italian: poor'	1	L P {M Sg B} St	524.25
6	117	Main discipline of study: commerce'	3	L P {M Sg B} St	372.91
12	47	Knowledge of Spanish: poor'	6	{L P} B {M Sg} St	51.10
24	38	Sex: male'	12	{L P} {B M Sg} St	32,91
48	21	Knowledge of English: poor'	24	P L Sg B {M St}	10.40
97	17	Intention of international degree: yes'	48	P L B Sg {M St}	6.25
49	17	Knowledge of English: poor'	24	L P M B {Sg St}	5.58
99	15	Intention of international degree: yes'	49	L P M {Sg B St}	4.53
13	70	Full-time employment: yes'	6	L P {M Sg B} St	142.17
27	64	Intention of international degree: yes'	13	L P Sg {M B St}	118.58
54	19	Knowledge of English: poor'	27	{L P} Sg {M B} St	10.34
109	16	Sex: male'	54	L P {M Sg B} St	6.86
55	45	Knowledge of English: poor'	27	L P Sg {M B St}	57.30
111	39	Sex: male'	55	L P Sg {M B St}	43.56
222	21	Knowledge of Spanish: poor'	111	L P Sg {M B St}	12.38
7	21	Knowledge of Spanish: poor'	3	L M P {Sg B} St	10.73
14	15	Main discipline of study: commerce'	7	{L M} P Sg B St	4.70
17	2	Terminal node	8	B L St {P M Sg}	0.07
67	7	Terminal node	33	L Sg B P St M	1.34
37	14	Terminal node	18	L P B {M St Sg}	4.78
39	8	Terminal node	19	L B {P Sg} {M St}	1.45
11	8	Terminal node	5	M {L Sg} {P B St}	1.13
25	9	Terminal node	12	P L B M {Sg St}	1.59
195	5	Terminal node	97	P L B {M Sg St}	0.50
199	8	Terminal node	99	{L P} {M B} {Sg St}	1.30
219	10	Terminal node	109	P L Sg {M B} St	2.52
223	18	Terminal node	111	L St {P Sg} {M B}	9.08
445	3	Terminal node	222	L Sg St P {M B}	0.16
15	6	Terminal node	7	B L P {M Sg} St	0.76
29	7	Terminal node	14	M L Sg P B St	0.88
32	1	Terminal node	16	Sg L St M M P	0.00
66	10	Terminal node	33	{L Sg} St {P M B}	2.52
36	4	Terminal node	18	L St B {P M Sg}	0.25
38	13	Terminal node	19	L B {Sg St} {P M}	5.29
10	7	Terminal node	5	L M P B {Sg St}	1.42
96	4	Terminal node	48	P {L Sg} M {B St}	0.39
194	12	Terminal node	97	P L Sg B M St	3.00
98	2	Terminal node	49	L P {M Sg B} St	0.01
198	7	Terminal node	99	P L M {B St} Sg	0.90
26	6	Terminal node	13	P {L M B} Sg St	0.80
108	3	Terminal node	54	{L Sg} {M B} P St	0.25
218	6	Terminal node	109	L P {M Sg B St}	0.83
110	6	Terminal node	55	L {P M} Sg {B St}	0.76
444	18	Terminal node	222	L P Sg {M B St}	9.13
28	8	Terminal node	14	{L P} M {Sg B St}	1.13

Table 5.5: DBMTR output

5.6 Concluding remarks

In this chapter a tree-based model dealing with preference rankings response matrix has been introduced.
Preference decisions usually depend on the characteristics of both the judges and the objects being judged. This theme has been handled in literature with log-linear representation of generalized Bradley-Terry model [34, 35].
This new approach is neither better nor worst than the other: it is just a new method. Of course there are several advantages in using tree-based methods: interpretation easiness, they provide an intuitive graphical representation, don't require the specification of a model structure, consider conditional interactions among variables.
As well known, tree-based models have two main aims: explanatory and confirmatory.
When they are used in the former sense, trees allow observing interactions between rankings and covariates, and catching the differences in preference for different groups of individuals. As confirmatory tools trees can be used to predict preference rankings.
There are several phases in tree growing:

- the definition of an impurity measure;
- the definition of a splitting criterion;
- the assignment of classes at each node;
- the definition of a prediction error.

As impurity measure the sum of Kemeny distance at the generic node was chosen. There are several distance models dealing with preference rankings [78, 62, 1], but we chose Kemeny distance because it is the unique metric which satisfies a set of axioms that a suitable distance measure for rankings should satisfy [67].

As a consequence, the minimization of the within-node Kemeny distance was chosen as splitting criterion.
We chose the consensus ranking as class-assignment rule, in the sense that the generic node is assigned to the ranking that represent the best representation of the overall set of preferences. The consensus ranking is defined as that ranking for which the sum between itself and the entire set of rankings is a minimum [67, 42].
Within the framework of classification and regression trees, the prediction error is defined as the misclassification ratio when referred to classification trees, whereas it coincides with the impurity when referred to regression trees. For the Distance-Based Multivariate Tree for Rankings the prediction error was defined as the sum of the diagonal elements of the distance sub-matrix between the i^{th} observed and fitted ranking (see equation 5.9).
Tree growing stops when a stopping rule occurs (bound on the decrease in impurity, bound on the number of observations, bound on the tree size).

New developments will be about DBMTR. In example the investigation of the prediction error definition to develop a pruning procedure, or a faster way to generating split. Indeed the second aim of a tree-based method (the confirmatory purpose) needs to be investigated. Moreover, the computational cost of the DBMTR is quite hard: recall that finding the consensus ranking is known to be a NP-hard problem. A binary tree with n terminal nodes has a total of $2n - 1$ nodes, as a consequence of the computation of $2n - 1$ consensus rankings. A way to define fastest splits according the FAST philosophy [84] has our highest research priority.

Conclusions

Tree-based models have been reaching a great interest in the scientific community, considering two main purposes: explanatory and confirmatory. With respect to the former, so far in literature partitioning procedures have been proposed taking into account different types of variables, such as nominal, ordinal and numerical. As a matter of fact, trees have been never dealt with preference rankings. This thesis has provided a suitable methodological approach to exploratory tree-based modeling preference ranking. As it concerns the confirmatory purpose, several methods based on decision trees have been proposed but their use to specific context applications require further investigation and methodological development. This is the case of statistical data editing, in particular missing data imputation and data fusion, for that suitable decision tree-based methods and algorithms have been provided.

As a result, this thesis has focalized the attention on two different frameworks: data editing (within the confirmatory approach) and preference rankings (within the exploratory approach).

As it concerns the first framework, trees have been recently proposed as nonparametric method for statistical data editing. The main goal of statistical data editing is to define improved procedures and greater automation to enhance the ability of survey managers and analysts to get published estimates. Data editing is a preliminary step of Knowledge Mining, such to obtain a database characterized by homogeneous, complete, coherent, and, in general, validated data from the quality point of view. In this context, the thesis has focalized the attention on *missing data imputation* and *data fusion* working on the definition of suitable methods based on decision trees in order to improve the accuracy of tree-based data imputation as well as to get the tree-based fusion of data sets discarding the nature of variables, respectively.

A general tree-based methodology for missing data imputation as well as specific algorithms to obtain the final estimates have been provided in this work. Following the incremental imputation philosophy based on cross-validated decision trees and a lexicographic ordering of the single data to be imputed, this work has considered an ensemble method characterized by boosting algorithms where tree-based models have been used as learner. Furthermore, the incremental imputation has concerned missing data of each variable at turn.
As a result, the BINPI algorithm (Boosted Incremental Non Parametric Imputation) has been developed: it uses a *STUMP* as weak learner when the variable under imputation is binary, whereas it uses a tree when the variable to be imputed is multi-class or numerical. The algorithm was tested on several simulated data sets as well as on a well known real data set from UCI machine learning repository: Boston housing. Both the simulations and the real data have shown the overall good performance of the proposed method against some classic competitors (standard parametric imputation tools, unconditional mean imputation).
Main results of the study can be summarized in this way:

- Imputation of a variable at turn is preferred to the imputation of a single data at turn;

- Boosting algorithms allow for a more accurate imputation;
- *STUMP* is ideal for a two-class problem in terms of computation efficiency;
- Fast tree is preferred to Stump for imputation of numerical or multi-class missing values in terms of accuracy.

It is working in progress a study of the back-propagation error of imputed data due to the incremental approach.

In chapter four, an innovative methodology for Data Fusion based on an incremental imputation algorithm in tree-based models has been provided. In particular, we have considered robust tree validation by boosting iterations.
Data Fusion can be considered as a special case of data imputation where the values to be imputed are those allowing the merging between two different sample surveys. The resulting developed algorithm, named Robust Tree-based Incremental Imputation algorithm (RTII), belongs to the explicit models family for data fusion. Several data sets were simulated to test how RTII algorithm works; as benchmarking methods both explicit (such as standard trees and multiple regression) and implicit methods (based on factorial techniques) were considered. The proposed methodology presents two special advantages:

- it can be considered for a mixed data structure that includes both numerical and categorical variables. In this case, factorial techniques cannot be suitably used, as well as classical regression models fail the goal of a unique multiple imputation of missing data;

- it allows to reconstruct the imputed variable distribution in terms of both mean and variance. A common problem concerning to classical imputation methods is to reduce, in a significant way, the variance of imputed distribution. Simulation studies show how all methods work well in mean reconstruction, whereas tree-based methods reconstruct better than other techniques the variability of the variable under imputation. The use of boosting algorithms get the imputation more accurate.

Data fusion is more ambitious than "simple" missing data imputation: values are missing because they never have been collected. The risk is to get banal imputations. For this reason, the study of pre-fusion conditions will be studied in the near future.

As it concerns the second framework, the Distance-Based Multivariate Tree for Rankings (DBMTR) introduced in this thesis is a new explanatory method to investigate which predictors and which interactions of predictors are the most significant to explain the response variable when this is constituted by rank order preference data or paired comparison rankings.
Preference rankings depend on the characteristics of both the judges and the objects being judged: in the literature this specific topic has been handled with log-linear models.
To grow a tree structure, both the definition of an impurity measure and a class-assignment rule in the nodes are necessary. Impurity chosen is the sum of Kemeny distance within node whereas the ranking-class assignment rule is the consensus ranking. The Kemeny distance satisfies a set of axioms that a suitable distance measure for rankings should satisfy:

- it is a metric, so that satisfies the well known properties of non negativity, symmetry and triangular inequality;

- the minimum positive distance is one;
- the measure of distance should not be affected by a relabeling of the set of objects to be ranked;
- if two rankings are in complete agreement at the beginning and at the end of the list and differ only in the middle, than the distance does not change after deleting these two rankings.

The consensus ranking is defined as that ranking for which the sum between itself and the entire set of rankings is a minimum.
These two measures are the foundation on which the resulting DBMTR is grown. This new approach is neither better nor worst than the log-linear models: it is just a new method. It is doubtless better interpretable.
Future perspectives about DBMTR concern the definition of fastest splits because of the hard computational cost in computing the consensus ranking. In addiction a suitable pruning procedure needs to be investigated to extend the DBMTR from the explanatory to the confirmatory point of view.

The scientific results of this work can be improved: the hope is that they have induced curiosity in the reader.

Appendix A

MatLab codes

A.1 AdaBoost

A.1.1 AdaBoost algorithm for binary classification problems

```
function [classifier,Btest_error,Etest,Etraining,test,training,E]=adaboost...
(trainingX,trainingY,testX,testY,k_max,sm)
%Esegue k_max iterazioni di boosting (Adaboost; weak learner CART)
% per problemi di classificazione, variabile risposta binaria.
%Parametri di input
%-TrainingX     Matrice dei predittori
%-TrainingY     Vettore delle variabili risposta
%-TestX         Matrice dei predittori da utilizzare come campione test
%-TestY         Vettore delle variabili risposta da associare a TestX quale campione
%test
%-K_max         Numero di iterazioni di boosting
%-sm            numerosità minima per poter splittare
%Output
%-classifier    classificatore aggregato
%-E_boo         Errore test del classificatore aggregato
%-E_iter        Andamento dell'errore (test) del classificatore aggregato attraverso le
%iterazioni
%-test          Stima del tasso di errata classificazione attraverso il campione test
%dell'albero singolo
%-E             Errore ponderato durante le iterazioni
%-E_learning    Andamento dell'errore di training del classificatore aggregato attraverso
%le iterazioni



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 6
    sm=5;
end
%Calcolo della stima dei tassi di malclassificazione dell'albero singolo per
%risostituzione e attraverso il test set

T_single=treefit(trainingX,trainingY,'method','c','splitmin',sm); %Albero max
[cc,ss,nn,bb]=treetest(T_single,'test',testX,testY);
tt=treeprune(T_single,'level',bb);  %pruning
Ctraining=treeval(T_single,trainingX);
Ctest=treeval(tt,testX);
training=sum(Ctraining~=trainingY)/size(Ctraining,1);  %errore di training albero singolo
test=sum(Ctest~=testY)/size(Ctest,1);   %errore test albero singolo
```

```
fullY=[trainingY;testY];
fullX=[trainingX;testX];
[r,c] = size(trainingX);
[ri,co]=size(testX);
%[rig,col]=size(fullX);
D=ones(1,r)/r;
D=D';
IterDisp=10;
Hx=zeros(size(fullX,1),1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%inizio della manipolazione del training set per il boosting
for k=1:k_max
   randnum = rand(1,r);
   cD = cumsum(D);
   indices = zeros(1,r);
    for t=1:r
        loc = max(find(randnum(t) > cD))+1;  %aumenta la probabilità di estrazione
         %delle osservazioni con i pesi più elevati
        if isempty(loc)
         indices(t) = 1;
      else
         indices(t) = loc; %indicizzazione del training set
      end
   end

   %Addestramento del weak learner
    T_boost = treefit(trainingX(indices,:),trainingY(indices,:),...
    'method','c','splitmin',sm); %Albero max
    Ctestboost=treeval(T_boost,fullX); %weak hypotesys


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    E(k) = sum(D.*(Ctestboost(1:r) ~= trainingY)); %Errore ponderato attraverso le
     %iterazioni

        if (E(k) == 0)
        disp('error equal to zero')
        break
    end

    alpha_k = 0.5*log((1-E(k))/E(k));  %Calcolo del parametro alpha


    D  = D.*exp(alpha_k*(Ctestboost(1:r)~=trainingY)*2-1); %Aggiornamento dei pesi
    D  = D./sum(D); %normalizzazione dei pesi
    %indicator=[Ctestboost==1 Ctestboost==2];
    Ctestb=Ctestboost-1;
    Hx=Hx+alpha_k*((2*Ctestb)-1);
    classifier=(Hx>0)+1;
    Etest(1,k)=mean(classifier(r+1:end)~=testY);  %errore classificatore boosted sul
     %campione test
    Etraining(1,k)=mean(classifier(1:r)~=trainingY); %errore classificatore boosted
     %sul campione training
    if (k/IterDisp == floor(k/IterDisp))
          disp(['Completed ' num2str(k) ' boosting iterations'])
    end
end
classifier=classifier(r+1:end);
Btest_error=Etest(end);
H=menu('Do you want plot the figure?','Yes','No')
if H==1
    plot([1:1:k_max],Etraining,'k',[1:1:k_max],Etest,'r',[1:1:k_max],...
    ones(1,k_max)*test,':')
    legend('training error AdaBoost','test error AdaBoost','test error single tree')
    xlabel('AdaBoost iterations')
    ylabel('Error')
end
```

A.1.2 AdaBoost algorithm for multiclass classification problems

```
function [classifier,Btest_error,Etest,Etraining,test,training,E] = ...
adaboostM(trainingX,trainingY,testX,testY,k_max,sm)
%%Algortimo AdaBoost per classificaioni multiclasse, versione di Eibl &
%%Pfeiffer
%Esegue k_max iterazioni di boosting (Adaboost; weak learner CART).
%Parametri di input
%-TrainingX     Matrice dei predittori
%-TrainingY     Vettore delle variabili risposta
%-TestX         Matrice dei predittori da utilizzare come campione test
%-TestY         Vettore delle variabili risposta da associare a TestX quale
%campione test
%-K_max         Numero di iterazioni di boosting
%-sm            numerosità minima per poter splittare
%Output
%-classifier    classificatore aggregato
%-Btest_error   Errore sul test set del classificatore aggregato
%-Etest         Andamento dell'errore (test) del classificatore aggregato
%attraverso le iterazioni
%-Etraining     Andamento dell'errore di training del classificatore aggregato
%attraverso le iterazioni
%-test          Stima del tasso di errata classificazione attraverso il campione
%test dell'albero singolo
%-training      errore sul training set dell'albero singolo
%-E             Errore ponderato durante le iterazioni



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Le classi G della variabile risposta devono essere codificate...
 come (1,...,g,...G)')

if nargin < 6
    sm=5;
end
%Calcolo della stima dei tassi di malclassificazione dell'albero singolo per
%risostituzione e attraverso il test set

T_single=treefit(trainingX,trainingY,'method','c','splitmin',sm); %Albero max
[cc,ss,nn,bb]=treetest(T_single,'test',testX,testY);
tt=treeprune(T_single,'level',bb);  %pruning
Ctraining=treeval(T_single,trainingX);
testset=treeval(tt,testX);
training=mean(Ctraining~=trainingY);  %errore di training albero singolo
test=mean(testset~=testY);   %errore test albero singolo
```

```
fullY=[trainingY;testY];
fullX=[trainingX;testX];
[r,c] = size(trainingX);
[ri,co]=size(testX);
%[rig,col]=size(fullX);
D=ones(1,r)/r;
D=D';
IterDisp=10;
classifier    = zeros(1, size(testY,1))';
targets_matrix=zeros(k_max,size(testY,1))';
learning_classifier=zeros(1,size(trainingY,1))';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
G=size(tabulate(trainingY),1);
g=tabulate(trainingY);
g=g(:,1);
%inizio della manipolazione del training set per il boosting
for k=1:k_max
   randnum = rand(1,r);
   cD = cumsum(D);
   indices = zeros(1,r);
    for t=1:r
        loc = max(find(randnum(t) > cD))+1;  %aumenta la probabilità di estrazione
        %delle osservazioni con i pesi più elevati
        if isempty(loc)
         indices(t) = 1;
      else
         indices(t) = loc; %indicizzazione del training set
      end
   end

   %Addestramento del weak learner
    T_boost = treefit(trainingX(indices,:),trainingY(indices,:),'method','c',...
    'splitmin',sm);
     %Albero max
    Ctestboost=treeval(T_boost,fullX);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    E(k) = sum(D.*(Ctestboost(1:r) ~= trainingY)); %Errore ponderato attraverso le
     %iterazioni

        if (E(k) == 0),
        disp('errore pari a zero')
        break
        end
    alpha_k=log(((G-1)*(1-E(k)))/E(k));  %Calcolo del parametro alpha


    D=D.*exp(-alpha_k*(Ctestboost(1:r)==trainingY));
    D=D./sum(D);

    conf(1,k)=alpha_k;
    Hx(:,k)=Ctestboost;
    for w=1:G
    Indicator=Hx==g(w);
    confidence(:,w)=Indicator*conf';
    end
    [maxc fullclassifier(:,k)]=max(confidence,[],2);
    Etraining(1,k)=mean(fullclassifier(1:r,end)~=trainingY);
     %errore boosted sul training set
    Etest(1,k)=mean(fullclassifier(r+1:end,end)~=testY);
     %errore boosted sul test set ad ogni iterazione
    if (k/IterDisp == floor(k/IterDisp)),
          disp(['Completate ' num2str(k) ' iterazioni di boosting'])
    end


end
classifier=fullclassifier(r+1:end,end);
Btest_error=Etest(end);


H=menu('Do you want plot the figure?','Yes','No')
if H==1
    plot([1:1:k_max],Etraining,'k',[1:1:k_max],Etest,'r',[1:1:k_max],...
    ones(1,k_max)*test,':')
    legend('training error AdaBoost','test error AdaBoost','test error single tree')
    xlabel('AdaBoost iterations')
    ylabel('Error')
end
```

A.1.3 AdaBoost algorithm for regression problem

```
function [Classifier,E_iter,E_test,Beta,E_te_s] = adaboostR(trainingX,trainingY,...
testX,testY,k_max,sm)

%Parametri di input
%-TrainingX     Matrice dei predittori
%-TrainingY     Vettore delle variabili risposta
%-TestX         Matrice dei predittori da utilizzare come campione test
%-TestY         Vettore delle variabili risposta da associare a TestX quale campione test
%-K_max         Numero di iterazioni di boosting
%-sm            numerosità minima per poter splittare
%Output
%-Classifier    classificatore aggregato
%-E_iter        Andamento dell'errore (test) del classificatore aggregato attraverso
%le iterazioni
%-E_test        Errore test del classificatore aggregato
%-E_te_s        Stima del tasso di errata classificazione attraverso il campione
%test dell'albero singolo
%-Beta          Andamento della "confidenza" della classificazione



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 6
    sm=round(0.05*size(trainingX,1));
end
%Calcolo della stima dei tassi di malclassificazione dell'albero singolo per
%risostituzione e attraverso il test set

T_single=treefit(trainingX,trainingY,'method','r','splitmin',sm); %Albero max
[cc,ss,nn,bb]=treetest(T_single,'test',testX,testY);
tt=treeprune(T_single,'level',bb);  %pruning
Ctraining=treeval(T_single,trainingX);
Ctest=treeval(tt,testX);
E_tr_s=sqrt(mean((Ctraining-trainingY).^2)); %RMSE errore training
E_te_s=sqrt(mean((Ctest-testY).^2)); %Rmse errore test

fullY=[trainingY;testY];
fullX=[trainingX;testX];
[r,c] = size(trainingX);
[ri,co]=size(testX);
[rig,col]=size(fullX);
D=ones(1,r)/r;
D=D';
IterDisp=10;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%inizio della manipolazione del training set per il boosting
for k=1:k_max
   randnum = rand(1,r);
   cD = cumsum(D);
   indices = zeros(1,r);
    for t=1:r
        loc = max(find(randnum(t) > cD))+1;  %aumenta la probabilità di estrazione
         %delle osservazioni con i pesi più elevati
        if isempty(loc)
         indices(t) = 1;
      else
         indices(t) = loc; %indicizzazione del training set
      end
   end

   %Addestramento del weak learner
    T_boost = treefit(trainingX(indices,:),trainingY(indices,:),...
    'method','r','splitmin',sm); %Albero max
    Ctestboost=treeval(T_boost,fullX);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    L=(Ctestboost(1:r)-trainingY).^2;
    %Vedi Gey-Poggi  Boosting and instability for regressio trees
    LMed=sum(L.*D); %errore medio
    Beta(k)=LMed/(max(L)-LMed); %parametro Beta di confodenza della regressione
    DK=L./max(L);
    Pesi(k)=log(1/Beta(k)); %peso che servirà a calcolare la mediana ponderata
%     Pesi=Pesi./sum(Pesi);
    learner(:,k) = Ctestboost; %stima del weak learner ala k-ma iterazione

    %% procedura di aggregazione%%%
    for qw=1:rig %media ponderata per ogni osservazione
        learning=learner(qw,:);
        [learning Indici]=sort(learning);
         %indicizzazione delle realizzazioni per ogni individuo
        G=[cumsum(Pesi(Indici));learning;Indici];
         %matrice (di controllo) dei pesi cumulati associati
         %ad ogni osservazione
        Ind2=find(G(1,:)>0.5*sum(Pesi));
         %cerca la posizione mediana ponderata
        Best=G(2,Ind2(1,1));
         %indice della mediana ponderata
        ClassTemp(qw,1)=Best;
         %miglior realizzazione per ogni osservazione al tempo k
    end
```

```
     %%Fine aggregazione

    Be=Beta(k).^(1-DK);
    D=D.*Be;  %Aggiornamento della distribuzione secondo i pesi
    D=D./sum(D); %Normalizzazione della distribuzione
    E_iter(k)=sqrt(mean((ClassTemp(1:r)-trainingY).^2));
     %Root mean squared error di learning durante le iterazioni
    E_test(k)= sqrt(mean((ClassTemp(r+1:end)-testY).^2));
     %Root mean squared error del campione test durante le iterazioni

    if (k/IterDisp == floor(k/IterDisp)),
          disp(['Completed ' num2str(k) ' boosting iterations'])
    end

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
Classifier = ClassTemp(r+1:end); %classificatore Boosted
Errore_Test = sqrt(mean((Classifier-testY).^2));


PP = menu('Vuoi visualizzare il grafico?','si','no')
if PP==1
    plot([1:1:k_max],E_iter,'b-',[1:1:k_max],E_test,'r-',[1:1:k_max],...
    [ones(1,k_max).*E_te_s],'k:')
    legend('Training error AdaBoost','Test error AdaBoost',...
    'Test error single tree')
    xlabel('AdaBoost replications')
    ylabel('Root Mean Squared Error')
end
```

A.2 BINPI algorithm

```
function newy=BINPI(X,tipoX,k_max,sm)

%INPUT
%X      matrice contenente dati mancanti
%TipoX  vettore indicante il tipo dele colonne di X (0 se nominali ordinabili,
%1 se nominali non ordinabili, 2 se numeriche)
%Metodo 1=Exploratory Tree, 2=Decision Tree (test set 33%)

if nargin < 4
    sm = 5;
end

mis=0;
ri=sum(X,2);
col=sum(X,1);
N = find(isnan(col));
NN = find(~isnan(col));
indcol=[NN N];
M = find(isnan(ri));
MM = find(~isnan(ri));
indri=[MM;M];
y=X(indri,indcol);
%matrice con permutazione righe e colonne senza ordine
%lessicografico(creazione y)
                  %      per esempio A=[1 2 3 3 3              y=[1 3 3 2 3
                  %                     2 X 1 1 4                 1 3 5 2 4
                  %                     5 3 2 X 2                 2 1 4 X 1
                  %                     3 X 4 5 2                 5 2 2 3 X
                  %                     4 2 3 X 6                 3 4 2 X 5
                  %                     1 2 3 4 5]                4 3 6 2 X]
newtipo=tipoX(indcol);
index1=find(newtipo==1);
P=size(N,2);
for k=1:P
    if tipoX(N(k))==2;
        disp('imputing numerical variable')
        [newy,y,miss]=adaboost_Reg_incrimp(y,index1,newtipo,k_max,sm);
    elseif tipoX(N(k))<2;
        if size(tabulate(X(:,N(k))),1)<3;
            disp('imputing binary variable')
            [newy,y,miss]=adaboost_incrimp(y,index1,newtipo,k_max,sm);
        else
            disp('imputing multiclass variable')
            [newy,y,miss]=adaboostM_incrimp(y,index1,newtipo,k_max,sm);
        end
    end
```

```
end
[kr,rig]=sort(indri);
[kc,colo]=sort(indcol);
%%per tornare indietro all'ordine della matrice iniziale X
newy=newy(rig,colo);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [newy,y,miss] = adaboost_incrimp(y,index,newtipo,k_max,sm)
nacol=sum(y,1);
ab=find(isnan(nacol));
abb = find(~isnan(nacol));
abindcol=[abb ab];
Y=y(:,1:ab(1));
nari=sum(Y,2);
bc = find(isnan(nari));
bcc = find(~isnan(nari));
abindri=[bcc;bc];
Y=Y(abindri,1:ab(1));


[nakr,narig]=sort(abindri);%per ritormare a matrice trasformata
[nakc,nacolo]=sort(abindcol);
indices=find(~isnan(sum(Y,2)));
indicetest=find(isnan(sum(Y,2)));
trainingX=Y(indices,1:(end-1));
testX=Y(indicetest:end,1:(end-1));
miss=sum(isnan(Y(indicetest:end,end)));
trainingY=Y(indices,end);
risp=newtipo(size(testX,2)+1);
tip=newtipo(1:size(trainingX,2));
index=find(tip==1);
fullY=[trainingY];
%fullX=[trainingX;Xsupp];
fullX=[trainingX;testX];
[r,c] = size(trainingX);
% [ri,co]=size(testX);
[rig,col]=size(fullX);
[rigY,colY]=size(fullY);
D=ones(1,r)/r;
D=D';
IterDisp=10;
Hx=zeros(size(fullX,1),1);
for k=1:k_max
   randnum = rand(1,r);
   cD = cumsum(D);
   indicesB = zeros(1,r);
    for t=1:r
        loc = max(find(randnum(t) > cD))+1;
        if isempty(loc)
         indicesB(t) = 1;
      else
         indicesB(t) = loc;
      end
   end
  T_boost = treefit(trainingX(indicesB,:),trainingY(indicesB,:),'method',...
  'c','splitmin',sm);
  Ctestboost=treeval(T_boost,fullX);
  E(k) = sum(D.*(Ctestboost(1:r) ~= trainingY));
  if (E(k) == 0)
        disp('errore pari a zero')
        break
  end
  alpha_k = 0.5*log((1-E(k))/E(k));
  D  = D.*exp(alpha_k*(Ctestboost(1:r)~=trainingY)*2-1);
  D  = D./sum(D); %normalizzazione dei pesi
  Ctestb=Ctestboost-1;
  Hx=Hx+alpha_k*((2*Ctestb)-1);
  classifier=(Hx>0)+1;
  if (k/IterDisp == floor(k/IterDisp)),
          disp(['Completate ' num2str(k) ' iterazioni di boosting'])
  end
end
Classifier=classifier(rigY+1:end);
FullX=[trainingX;testX];
FullY=[trainingY;Classifier];
matricecomp=[FullX FullY];

YY=matricecomp(narig,:); %matrice completa e riversa (STEP2)
y=[YY y(:,size(YY,2)+1:end)]; %deve andare in input
newy=y;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [newy,y,miss] = adaboostM_incrimp(y,index,newtipo,k_max,sm)


nacol=sum(y,1);
ab=find(isnan(nacol));
abb = find(~isnan(nacol));
abindcol=[abb ab];
Y=y(:,1:ab(1));
nari=sum(Y,2);
bc = find(isnan(nari));
bcc = find(~isnan(nari));
abindri=[bcc;bc];
```

```
Y=Y(abindri,1:ab(1));


[nakr,narig]=sort(abindri);
[nakc,nacolo]=sort(abindcol);
indices=find(~isnan(sum(Y,2)));
indicetest=find(isnan(sum(Y,2)));
trainingX=Y(indices,1:(end-1));
testX=Y(indicetest:end,1:(end-1));
miss=sum(isnan(Y(indicetest:end,end)));
trainingY=Y(indices,end);
risp=newtipo(size(testX,2)+1);
tip=newtipo(1:size(trainingX,2));
index=find(tip==1);
fullY=[trainingY];
fullX=[trainingX;testX];
[r,c] = size(trainingX);
[rig,col]=size(fullX);
[rigY,colY]=size(fullY);
D=ones(1,r)/r;
D=D';
IterDisp=10;
G=size(tabulate(trainingY),1);
g=tabulate(trainingY);
g=g(:,1);
for k=1:k_max
   randnum = rand(1,r);
   cD = cumsum(D);
   indicesB = zeros(1,r);
    for t=1:r
        loc = max(find(randnum(t) > cD))+1;
        if isempty(loc)
         indicesB(t) = 1;
      else
         indicesB(t) = loc; %indicizzazione del training set
      end
   end


    T_boost = treefit(trainingX(indicesB,:),trainingY(indicesB,:),'method',...
    'c','splitmin',sm);
    Ctestboost=treeval(T_boost,fullX);
    E(k) = sum(D.*(Ctestboost(1:r) ~= trainingY));

        if (E(k) == 0),
        disp('errore pari a zero')
        break
        end
    alpha_k=log(((G-1)*(1-E(k)))/E(k));


    D=D.*exp(-alpha_k*(Ctestboost(1:r)==trainingY));
    D=D./sum(D);

    conf(1,k)=alpha_k;
    Hx(:,k)=Ctestboost;
    for w=1:G
    Indicator=Hx==g(w);
    confidence(:,w)=Indicator*conf';
    end
    [maxc fullclassifier(:,k)]=max(confidence,[],2);
    if (k/IterDisp == floor(k/IterDisp)),
          disp(['Completate ' num2str(k) ' iterazioni di boosting'])
    end
end
Classifier=fullclassifier(rigY+1:end,end);
FullX=[trainingX;testX];
FullY=[trainingY;Classifier];
matricecomp=[FullX FullY];

YY=matricecomp(narig,:);
y=[YY y(:,size(YY,2)+1:end)];
newy=y;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [newy,y,miss] = adaboost_Reg_incrimp(y,index,newtipo,k_max,sm)
nacol=sum(y,1);
ab=find(isnan(nacol));
abb = find(~isnan(nacol));
abindcol=[abb ab];
Y=y(:,1:ab(1));
nari=sum(Y,2);
bc = find(isnan(nari));
bcc = find(~isnan(nari));
abindri=[bcc;bc];
Y=Y(abindri,1:ab(1));
[nakr,narig]=sort(abindri);
[nakc,nacolo]=sort(abindcol);
indices=find(~isnan(sum(Y,2)));
indicetest=find(isnan(sum(Y,2)));
trainingX=Y(indices,1:(end-1));
testX=Y(indicetest:end,1:(end-1));
miss=sum(isnan(Y(indicetest:end,end)));
trainingY=Y(indices,end);
```

```
risp=newtipo(size(testX,2)+1);
tip=newtipo(1:size(trainingX,2));
index=find(tip==1);
fullY=[trainingY];
fullX=[trainingX;testX];
[r,c] = size(trainingX);
[rig,col]=size(fullX);
[rigY,colY]=size(fullY);
D=ones(1,r)/r;
D=D';
IterDisp=10;
inizio della manipolazione del training set per il boosting
for k=1:k_max
   randnum = rand(1,r);
   cD = cumsum(D);
   indicesB = zeros(1,r);
    for t=1:r
        loc = max(find(randnum(t) > cD))+1;
        if isempty(loc)
         indicesB(t) = 1;
      else
         indicesB(t) = loc;
      end
   end
    L=(Ctestboost(1:r)-trainingY).^2;
    LMed=sum(L.*D);
    Beta(k)=LMed/(max(L)-LMed);
    DK=L./max(L);
    Pesi(k)=log(1/Beta(k));
    learner(:,k) = Ctestboost;
    for qw=1:rig
        learning=learner(qw,:);
        [learning Indici]=sort(learning);
        G=[cumsum(Pesi(Indici));learning;Indici];
        Ind2=find(G(1,:)>0.5*sum(Pesi));
        Best=G(2,Ind2(1,1));
        ClassTemp(qw,1)=Best;
    end

    Be=Beta(k).^(1-DK);
    D=D.*Be;  %Aggiornamento della distribuzione secondo i pesi
    D=D./sum(D); %Normalizzazione della distribuzione
   if (k/IterDisp == floor(k/IterDisp)),
          disp(['Completate ' num2str(k) ' iterazioni di boosting'])
    end
end
Classifier = ClassTemp(rigY+1:end); %classificatore Boosted
FullX=[trainingX;testX];
```

```
FullY=[trainingY;Classifier];
matricecomp=[FullX FullY];
YY=matricecomp(narig,:); %matrice completa e riversa
y=[YY y(:,size(YY,2)+1:end)]; %deve andare in input
newy=y;
```

A.3 RTII algorithm

```
function [fit,E_iter,E_test]=FRTII(X,Y,X1,Y1,k_max,sm,type,...
catidx);
%%INPUT
%X X1: common variables
%Y: specific variables
%Y1: Control variables
%k_max: boosting iterations
%sm: minimum size at node to split
%type: column to be imputed type: 0=categorical, 1=numerical
%catidx: index of nominal categorical variable in X matrix
%%OUTPUT
%fit: fitted values
%E_iter: Training errors through boosting iterations
%E_test: Test errors through boosting iterations



[r c]=size(X);
[ri co]=size(X1);
[rY cY]=size(Y);
fullX=[X;X1];
[rig,col]=size(fullX);
if nargin < 8
    catidx=[]
end

for cx=1:cY %for1
    disp(['Imputation column number ' num2str(cx)])
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if type(1,cx)==1 %%%Regression%%%%%
        disp(['Fusion on numerical variable'])
        D=ones(1,r)/r;
        D=D';
        IterDisp=10;
        T_single=treefit(X,Y(:,cx),'method','r','splitmin',sm);
        [cc,ss,nn,bb]=treetest(T_single,'test',X1,Y1(:,cx));
        tt=treeprune(T_single,'level',bb);  %pruning
        Ctest(:,cx)=treeval(tt,X1);
        E_te_s(cx)=sqrt(mean((Ctest(:,cx)-Y1(:,cx)).^2));

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %inizio della manipolazione del training set per il boosting
        for k=1:k_max %for2
            randnum = rand(1,r);
            cD = cumsum(D);
            indices = zeros(1,r);
                for t=1:r %for3
                    loc = max(find(randnum(t) > cD))+1;
                    if isempty(loc)
                        indices(t) = 1;
                    else
                        indices(t) = loc; %indicizzazione del training set
                    end
                end %end for3

            T_boost = treefit(X(indices,:),Y(indices,cx),'method','r','splitmin',...
            sm);
            Ctestboost=treeval(T_boost,fullX);
            L=(Ctestboost(1:r)-Y(:,cx)).^2;
            LMed=sum(L.*D); %errore medio
            Beta(k)=LMed/(max(L)-LMed);
            DK=L./max(L);
            Pesi(k)=log(1/Beta(k));
            learner(:,k) = Ctestboost;

            for qw=1:rig  %%for4
                learning=learner(qw,:);
                [learning Indici]=sort(learning);
                G=[cumsum(Pesi(Indici));learning;Indici];
                Ind2=find(G(1,:)>0.5*sum(Pesi));
                Best=G(2,Ind2(1,1));
                ClassTemp(qw,1)=Best;
            end %end for4

            Be=Beta(k).^(1-DK);
            D=D.*Be;  %Aggiornamento della distribuzione secondo i pesi
            D=D./sum(D); %Normalizzazione della distribuzione
            E_Iter(k)=sqrt(mean((ClassTemp(1:r)-Y(:,cx)).^2));
            E_Test(k)= sqrt(mean((ClassTemp(r+1:end)-Y1(:,cx)).^2));
            if (k/IterDisp == floor(k/IterDisp)),
                disp(['Completed ' num2str(k) ' boosting itrations'])
            end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        end %end for2
        Classifier = ClassTemp(r+1:end);
        fit(:,cx)=Classifier;
        E_iter(:,cx)=E_Iter';
        E_test(:,cx)=E_Test';
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    elseif type(1,cx)==0
        if size(tabulate(Y(:,cx)),1)==2 %classification with 2 classes
            disp(['Fusion on binary variable'])
```

```
D=ones(r,1)/r;
D=D./sum(D);
IterDisp=10;
Hx=zeros(size(fullX,1),1);
T_single=treefit(X,Y(:,cx),'method','c','splitmin',sm);
[cc,ss,nn,bb]=treetest(T_single,'test',X1,Y1(:,cx));
tt=treeprune(T_single,'level',bb);  %pruning
Ctest(:,cx)=treeval(tt,X1);
E_te_s(cx)=1-mean(Ctest(:,cx)==Y1(:,cx));
for k=1:k_max  %beginning boosting
    randnum = rand(1,r);
    cD = cumsum(D);
    indicesB = zeros(1,r);
    for t=1:r
        loc = max(find(randnum(t) > cD))+1;
        if isempty(loc)
           indicesB(t) = 1;
        else
            indicesB(t) = loc;
        end
    end

T_boost = treefit(X(indicesB,:),Y(indicesB,cx),'method','c',...
'splitmin',sm);
Ctestboost=treeval(T_boost,fullX);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

E(k) = sum(D.*(Ctestboost(1:r) ~= Y(:,cx)));

if (E(k) == 0),
    disp('errore pari a zero')
    break
end

alpha_k = 0.5*log((1-E(k))/E(k));
D  = D.*exp(alpha_k*(Ctestboost(1:r)~=Y(:,cx))*2-1);
D  = D./sum(D); %normalizzazione dei pesi
Ctestb=Ctestboost-1;
Hx=Hx+alpha_k*((2*Ctestb)-1);
classifier=(Hx>0)+1;
E_Iter(k)=1-mean(classifier(1:r)==Y(:,cx));
E_Test(k)=1-mean(classifier(r+1:end)==Y1(:,cx));
if (k/IterDisp == floor(k/IterDisp)),
    disp(['Completed ' num2str(k) ' boosting iterations'])
end
```

```
        end %end boosting
        Classifier=classifier(r+1:end);
        fit(:,cx)=Classifier;
        E_iter(:,cx)=E_Iter';
        E_test(:,cx)=E_Test';
    else   %%Multiclass classification
        disp(['Fusion on multiclass variable'])
        D=ones(r,1)/r;
        D=D./sum(D);
        IterDisp=10;
        G=size(tabulate(Y(:,cx)),1);
        g=tabulate(Y(:,cx));
        g=g(:,1);
        T_single=treefit(X,Y(:,cx),'method','c','splitmin',sm);
        [cc,ss,nn,bb]=treetest(T_single,'test',X1,Y1(:,cx));
        tt=treeprune(T_single,'level',bb);  %pruning
        Ctest(:,cx)=treeval(tt,X1);
        E_te_s(cx)=1-mean(Ctest(:,cx)==Y1(:,cx));
        for k=1:k_max  %boosting begin
            randnum = rand(1,r);
            cD = cumsum(D);
            indicesB = zeros(1,r);
            for t=1:r
                loc = max(find(randnum(t) > cD))+1;
                if isempty(loc)
                    indicesB(t) = 1;
                else
                indicesB(t) = loc; %indicizzazione del training set
                end
            end
            T_boost = treefit(X(indicesB,:),Y(indicesB,cx),'method','c',...
            'splitmin',sm);
            Ctestboost=treeval(T_boost,fullX);
            E(k) = sum(D.*(Ctestboost(1:r) ~= Y(:,cx)));

            if (E(k) == 0),
                disp('errore pari a zero')
                break
            end
            alpha_k=log(((G-1)*(1-E(k)))/E(k));
            D=D.*exp(-alpha_k*(Ctestboost(1:r)==Y(:,cx)));
            D=D./sum(D);
            conf(1,k)=alpha_k;
            Hx(:,k)=Ctestboost;
            for w=1:G
                Indicator=Hx==g(w);
                confidence(:,w)=Indicator*conf';
                end
                [maxc fullclassifier(:,k)]=max(confidence,[],2);
                E_Iter(k)=mean(fullclassifier(1:r,end)~=Y(:,cx));
                E_Test(k)=mean(fullclassifier(r+1:end,end)~=Y1(:,cx));

                if (k/IterDisp == floor(k/IterDisp)),
                    disp(['Completed ' num2str(k) ' boosting iterations'])
                end
            end %boosting end
            Classifier=fullclassifier(r+1:end,end);
            fit(:,cx)=Classifier;
            E_iter(:,cx)=E_Iter';
            E_test(:,cx)=E_Test';
        end %elseif end
    end

end %for1 end
```

A.4 Distance-Based Multivariate Trees for Rankings

```
function [tree matrix Matrix sintfather sintchildren Imptree imp sintesi2...
 trees]=ranktree(X,Y,num,decrmin,indnom,graph)
%%%Distance-based Multivariate tree for Rankings using Kemeny Distance
%X -> Predictors matrix
%Y -> Response variables matrix
%num -> Minimum size at node to split
%decrmin -> Minimum impurity decreasing
%indnom -> Index of nominal variables; 0 = ordinal, 1 = nominal
%(i.e. [0 1 1 0 0 0 1])
%graph -> 0 = shows figure of error rate; 1 = no figure
%%%%%Output
%tree -> struct array with description of MRT at each node
%matrix -> block final matrix: [Number terminal node, X, Y]
%Matrix -> block final matrix:[Id individual, number terminal
%node, Y, X]
%sintfather -> struct array with description of father nodes
%sintchildren -> synthesis of children nodes for the left and right
%branch of the tree: [node number, size at node, imputed value,
%impurity at node]
%Imptree -> Overall impurity of the tree
%imp ->struct array showing the decrease of impurity at
%each node which generates a split
%sintesi2 -> synthesis of all terminal nodes:
%[node number, size at node, fitted value at node,
%impurity at node(last column)]
%trees -> struct array: contains all subtrees for pruning procedure
%%%%%%%%

%%%Antonio D'Ambrosio; Leiden, 2007 (February - April)
%%%%%If we need more memory, use this code below
% cwd=pwd;
% cd('F:\temp');
% pack;
% cd(cwd);
%%%%%End code for more memory
tree.indnom=indnom;
cicciput=X;
[N C]=size(Y);
[r c]=size(X);
id=(1:1:N)';
it=0;it2=0;
tree.nodo(1).X=X;
tree.nodo(1).Y=Y;
tree.nodo(1).term=0;
tree.nodo(1).father=0;
tree.nodo(1).n=size(X,1);
tree.nodo(1).impur=rankimpurity(Y);
[Cons Err]=consrank(Y);
tree.nodo(1).class=Cons;
tree.nodo(1).error=Err;
tree.fiteval=Y;
tree.num=num;
tree.decrmin=decrmin;
%tree.typeimpurity=impurity;
L=1;
cont=0;
imp.decimpurita=0;
imp.nodo=0;
memnodo=[0,1];
lung=length(memnodo);
nodo1(1:size(Y,1),1)=1;
matrice=[nodo1(:,1) X Y];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%splitting and stopping rules
while memnodo(lung) ~= 0
    it=it+1;
    while size(X,1)>=num
        [XL XR YL YR indpred valsplit Impadre decr synt errornode ...
        ImpL ImpR slvalue vsplit Valsplit]=ranksplit(X,Y,N,indnom);
        if decr <= decrmin
            if isempty(YL) | isempty(YR)
                tree.nodo(L).decrimp=[];
            else
                tree.nodo(L).decrimp=decr;
            end
            tree.nodo(L).term=1;
            break
        end
%         0;
%         if isempty(YL) | isempty(YR)
%             tree.nodo(L).decrimp=[];
%             tree.nodo(L).term=1;
%             break
%         end
%
        cont=cont+1;

        sintfather.numnode(cont)=L;
        sintfather.varsplit(cont)=valsplit;   %variable which generates
        %the split and rule of the split
        sintfather.sizenode(cont)=length(X(:,1));
        sintfather.impurity(cont)=Impadre;


        if indnom(1,indpred)==0  % ordinal case
            indiceL=find((matrice(:,indpred+1)<=vsplit)&...
            (matrice(:,1)==L));
            matrice(indiceL,1)=L*2;
            indiceR=find(matrice(:,1)==L);
            matrice(indiceR,1)=L*2+1;
        else   %nominal case
            for ii=1:length(slvalue)
                indiceL=find((matrice(:,indpred+1)==slvalue(ii))...
                & (matrice(:,1)==L));
                matrice(indiceL,1)=L*2;
            end
            indiceR=find(matrice(:,1)==L);
            matrice(indiceR,1)=L*2+1;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%end splitting and stopping rules

        L=L*2;
        R=L+1;
        tree.nodo(L).X=XL;
        tree.nodo(R).X=XR;
        tree.nodo(L).Y=YL;
        tree.nodo(R).Y=YR;
        tree.nodo(L/2).term=0;
        tree.nodo(L).father=L/2;
        tree.nodo(R).father=L/2;
        tree.nodo(L/2).nom=indnom(indpred);
        tree.nodo(L/2).split=slvalue; %categories belonging
         %to splitting rule
        tree.nodo(L/2).col=indpred;
        tree.nodo(L/2).valsplit=valsplit;  %%splitting rule
        tree.nodo(L).impur=ImpL;
        tree.nodo(R).impur=ImpR;
        tree.nodo(L/2).decrimp=decr;
        tree.nodo(L).class=synt(1,2:end-1);
        tree.nodo(R).class=synt(2,2:end-1);
        tree.nodo(L).n=size(XL,1);
        tree.nodo(R).n=size(XR,1);
        tree.nodo(L).number=L;
        tree.nodo(R).number=R;
        tree.nodo(L).error=errornode.L;
        tree.nodo(R).error=errornode.R;

        sintchildren.R(cont,:)=[R synt(2,:)]; %node number,
         %size at node,
         %rankings,
         %impurity at node
        sintchildren.L(cont,:)=[L synt(1,:)];

        memnodo=[memnodo,R,L];
        X=tree.nodo(L).X;
        Y=tree.nodo(L).Y;
        lung=length(memnodo);
        memnodo(lung)=[];



        it2=it2+1;
        imp.decimpurita(it2)=decr;
        imp.nodo(it2)=(L/2);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%
    end %end del secondo while
    if size(X(:,1)) < num
        tree.nodo(L).term=1;
    end
    lung=length(memnodo);
    L=memnodo(lung);

    if size(X(:,1)) < num | decr < decrmin
        memnodo(lung)=[];
        lung=length(memnodo);
    end


    if L > 1
       X=tree.nodo(L).X;
       Y=tree.nodo(L).Y;
    end
end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if tree.nodo(1).term==1
    noditot=0;
    noditot2=0;
    sintchldren=0;
    sintfather=0;
    sintesi=0;
    sintesi2=0;
    decr=0;
    matrix=0;
else
    noditot=[sintchildren.R(:,1:2) ; sintchildren.L(:,1:2)];
```

```
     %node number and size
    noditot2=[sintchildren.R; sintchildren.L];
    noterm=sintfather.numnode';
    noterm(1)=[];
    noterm;
    n=length(noterm);
    m=length(noditot(:,1));
    cont=0;

%% Terminal nodes listing

    for j=1:m
        term=1;
        for i=1:n
            if noterm(i) == noditot(j,1)
                term=0;
                i=n;
            end
        end
        if term==1
            cont=cont+1;
            sintesi(cont,1:2)=noditot(j,1:2); % Terminal nodes
             %and their size
            sintesi2(cont,:)=noditot2(j,:); %nTerminal node number,
             %frequence, rankings and
             %impurity
        end
    end
    matrix=matrice; %Terminal node nmber, then X matrix and Y matrix
end
Matrix=[id matrix(:,1) matrix(:,c+2:end) matrix(:,2:c+1)];
%id individual, Node number, Y and X matrices
fit=zeros(N,C+1);
valorimedi=sintesi2(:,3:end-1);
valorimedi=[valorimedi sintesi2(:,1)];
for g=1:size(sintesi2,1);
    igg=find(Matrix(:,2)==sintesi2(g,1));
    for w=1:C+1
        fit(igg,w)=valorimedi(g,w);
        0;
    end
end
fitm=[id fit];    %id, fit(for all Y variable),
%terminal node number (
%in the last column)
[trees,seq,nodint,alfa,prune]=sequenzeMT(tree,sintfather...
,sintchildren,sintesi2);
tree.alpha=[0 alfa];
tree.prunenode=prune;
tree.pruningseq=seq;
tree.fitvalue=fitm;




[aa bb]=size(tree.nodo);
for p=1:bb;
    if tree.nodo(p).term==1
        treeerr(p,1)=tree.nodo(p).error;
        treeimp(p,1)=tree.nodo(p).impur;
    else
        treeimp(p,1)=0;
        treeerr(p,1)=0;
    end
end
Imptree.imp=sum(treeimp);
Imptree.error=sum(treeerr);
if graph==0
    ME=menu('Do you want see learning error rate progress...
     through sub-tree sequence?','yes','no');
    if ME==1
        evaltreeMT(cicciput,tree,trees);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [XL XR YL YR indpred valsplit Impadre decr synt errornode ImpL...
 ImpR slvalue vsplit Valsplit]=ranksplit(X,Y,N,indnom);
%X -> Matrix of predictors
%Y -> Matrix or rensponse variables
%indnom -> index of nominal and ordinal variables
%%Output:
%XR -> X matrix on the right node; YR -> Y matrix on the right node
%XL -> X matrix on the left node;  YL -> Y matrix on the left node
%indpred -> Variable belonging to X matrix which generates the split
%valsplit -> splitting rule
%decr -> Maximum impurity decreasing
%ImpL -> Impurity at left node
%ImpR -> Impurity at right node

id=(1:1:size(Y,1))';
[r c]=size(X);
for k=1:c;
    h=tabulate(X(:,k));
    h=h(:,1);
    [rh ch]=size(h);
    0;
    if rh == 1
        for s=1:rh
            Decrementi(s,k)=0;
            modr(s)=h(s);
        end %end3
    else
        if indnom(1,k)==1
            [Xleft,Yleft,Xright,Yright,Bestdecr,indu,combsplit]=...
            nominalsplit(X,Y,k,Impadre,N);
            BestDecrXs(k)=Bestdecr;
            Decrementi(1,k)=Bestdecr;
            ind(k)=indu;
            value(1,k)=nan;
            spvalue(k).split=combsplit.L;
            Valsplit(k).split={['X' num2str(k) ' ==...
             ' num2str(combsplit.L)]};
            Left(k).Y=Yleft;
            Left(k).X=Xleft;
            Right(k).Y=Yright;
            Right(k).X=Xright;
        else
            for s=1:rh-1;
                iL=find(X(:,k)<=h(s));
                 %Index of categories of X distribution to send to left
                 Yleft=Y(iL,:);             %Y distribution on the left
                 nL=size(Yleft,1);
                 GL=rankimpurity(Yleft);
                 iR=find(X(:,k)>h(s));
                  %Index of categories of X distribution to send to right
                 Yright=Y(iR,:);            %Y distribution on the right
                 nR=size(Yright,1);
                 GR=rankimpurity(Yright);
                 Decrementi(s,k)=Impadre-((GL/N)+(GR/N));
                 modr(s)=h(s);
            end %end del for s=1:rh-1;
            [BestDecrXs(k) ind(k)]=max(Decrementi(:,k));
            value(1,k)=modr(ind(k));
            spvalue(k).split=modr(ind(k));
            Valsplit(k).split={['X' num2str(k) ' <= ' num2str(value(1,k))]};
        end %end del if indnom(1,k)==1
    end %end del if rh==1
end
if sum(Decrementi)==0   %if 1
    tabella = zeros(3,c);
    XL=[];
    XR=[];
    YL=[];
    YR=[];
    indpred=[];
    valsplit=[];
    Impadre=[];
    decr=0;
    synt=[];
    errornode=[];
    ImpL=0;
    ImpR=0;
    slvalue=[];
    vsplit=[];
    Valsplit=[];
else
    tabella=[BestDecrXs;ind;value];
    [decr indpred]=max(tabella(1,:));
%%%%%%%%%%%%%
    valsplit=Valsplit(indpred).split;
    slvalue=spvalue(indpred).split;
    if isnegative(decr)==1  %if 2
        disp('warning')
        decr=0;
        XL=[];
        XR=[];
        YL=[];
        YR=[];
        indpred=[];
        valsplit=[];
        Impadre=[];
        decr=0;
        synt=[];
        errornode=[];
        ImpL=[];
        ImpR=[];
        slvalue=[];
        vsplit=[];
        Valsplit=[];
    else
        vsplit=[];
        if indnom(1,indpred)==0  %if 3
            vsplit=tabella(3,indpred);
            XL=X(find(X(:,indpred)<=vsplit),:);
            XR=X(find(X(:,indpred)>vsplit),:);
            YL=Y(find(X(:,indpred)<=vsplit),:);
            YR=Y(find(X(:,indpred)>vsplit),:);
        else
            XL=Left(indpred).X;
            YL=Left(indpred).Y;
            XR=Right(indpred).X;
            YR=Right(indpred).Y;
        end  %end if 3
        if isempty(YL) | isempty(YR)  %if 4 %if matrix cannot be splitted
            %disp('non posso splittare')
            decr=0;
            synt=[];
            errornode=[];
            ImpL=[];
            ImpR=[];
            Impadre=[];
            valsplit=[];
            indpred=[];
        else

            ImpL=rankimpurity(YL)*size(YL,1)/N;
            ImpR=rankimpurity(YR)*size(YR,1)/N;
            [consL errorL]=consrank(YL);
            [consR errorR]=consrank(YR);
            errornode.L=errorL*size(YL,1)/N; %risk (or error) at left node
            errornode.R=errorR*size(YR,1)/N; %risk (or error) at rigth node
            synt(1,:)=[size(YL,1) consL ImpL];  %synthesis of left side of
             %matrix Y [size of part, mean
             %of Y, Impurity at node]
            synt(2,:)=[size(YR,1) consR ImpR];  %synthesis of rigth side of
             %matrix Y [size of part, mean
             %of Y, Impurity at node]
        end  %end if 4
    end  %end if 2
end  %end if 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Xleft,Yleft,Xright,Yright,BestDecr,ind,combsplit]=...
nominalsplit(X,Y,indnorm,Impadre,N);
tabXi=tabulate(X(:,indnorm));
indice=find(tabXi(:,2)==0);
tabXi(indice,:)=[];
XX=[1:length(tabXi)];
[comb,numcomb]=splitcomb(tabXi(:,1));
for i=1:numcomb
    indice=[];
    for j=1:length(comb.split(i).L)
        combsplit=tabXi(comb.split(i).L',1);
        indiceL=find(X(:,indnorm)==combsplit(j));
        indice=[indice; indiceL];
    end
    Xright=X(indice,:);
    Yright=Y(indice,:);
    nR=size(Yright,1);
    Xleft=X;
    Xleft(indice,:)=[];
    Yleft=Y;
    Yleft(indice,:)=[];
    nL=size(Yleft,1);
    GL=rankimpurity(Yleft);
    GR=rankimpurity(Yright);
    splitnum.split(i).XL=Xleft;
    splitnum.split(i).XR=Xright;
    splitnum.split(i).YL=Yleft;
    splitnum.split(i).YR=Yright;
    splitnum.split(i).R=tabXi(comb.split(i).L',1);
    splitnum.split(i).L=tabXi(comb.split(i).R',1);
    Decrementi(i,1)=Impadre-((GL/N)+(GR/N));
end
[BestDecr ind]=max(Decrementi);
0;


Xleft=splitnum.split(ind).XL;
Yleft=splitnum.split(ind).YL;
Xright=splitnum.split(ind).XR;
Yright=splitnum.split(ind).YR;
combsplit.L=splitnum.split(ind).L';
combsplit.R=splitnum.split(ind).R';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function SS=rankimpurity(Y);

[r c]=size(Y);
SS=kemenyd(Y);
SS=SS/r;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [kemeny kemenysq kem Y M]=kemenyd(X);
%%%kemeny = sum of kemeny distances
%%% kemenysq = sum of squared kemeny distances
[r c]=size(X);
M=kemenymatrix(X);
M=M*1;
```

```
kem=pdist(M,'cityblock');
kemeny=sum(kem);
%kemeny=sum(kem);
kemenysq=sum(kem.^2);
Y=squareform(kem);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function M=kemenymatrix(X);
%%Design matrix to computing Kemeny distance

[r c]=size(X);
column=combntns(1:c,2); %-> combination of pairwise
%comparison among abjects
c=size(column,1);

for k=1:r
    for j=1:c
        M(k,j)=sign((X(k,(column(j,1)))-X(k,(column(j,2)))));
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [consensus error cij]=consrank(X)

%%%%%%according Emond & Mason, 2002%%%
%%%% The higher is the number in X matrix
%%%% the more the object is preferred


[M N]=size(X);
if M==1
    consensus=X;
    error=0;
    cij=0;
else
    [kemeny kemenysq kemnorm kem t]=kemenyd(X);
    if sum(t,2) ~= 0
        wk=1./sum(t,2);
        wk=wk./sum(wk,1); %the lower is the kemeny distance,
         %the higher is the weight
    else
        wk=ones(M,1);
    end
    wk=ones(M,1);
    cij=zeros(N);
    for k=1:M
        s=squareform(scorematrix(X(k,:)));
        sl=triu(s);
        sl=sl+sl'*-1;
        a=find(abs(sl)==(N*(N-1))+1);
        if ~isempty(a)
            sl(a)=1;
        end
%     CIM(k).aij=sl;
       cij=cij+(wk(k)*sl);
    end
    candidate=findconsensus(cij);
    [kemeny kemenysq kemnorm kem t]=kemenyd([X;mean(X);...
    candidate]);
    somma=sum(t(M+1:end,1:M),2);
    [error b]=min(somma);
    error=error/M;
    if b==2 | somma(1,1)-somma(2,1)==0;%b(1,1)==b(2,1)
        consensus=candidate;
    else
        consensus=mean(X);
        disp('consenso medio')
    end
end

% si=squareform(scorematrix(meanr));
% sil=triu(si);
% sij=sil+sil'*-1;
% a=find(abs(sil)==(N*(N-1))+1);
% if ~isempty(a)
%     sij(a)=1;
% end
% V=sum(sum(abs(cij)));
% B=sum(sum(sij.*cij));
% P=V-B;




function Y=scorematrix(X);
%%Design matrix

[r N]=size(X);
column=combntns(1:N,2); %-> combination of pairwise
%comparison among abjects
c=size(column,1);
```

```
for k=1:r
    for j=1:c
          Y(k,j)=sign((X(k,(column(j,1)))-X(k,(column(j,2)))));
          if Y(k,j)==0
              Y(k,j)=N*(N-1)+1;
          end
    end
end


function X=findconsensus(cij)
X=ones(1,size(cij,2));
[M N]=size(X);
indici=combntns(1:N,2);
for j=1:size(indici,1)
        if (sign(cij(indici(j,1),indici(j,2)))>=1 & sign(cij(indici(j,2)...
        ,indici(j,1)))==-1)  %%if object i is ranked ahead of ogject j
            X(indici(j,1))=X(indici(j,1))+1;
            X(indici(j,2))=X(indici(j,2))+1;
        end
end
```

Bibliography

[1] Agresti A. (2002). *Categorical Data Ananlysis.* J. Wiley.

[2] Aluja-Banet T., Morineau A., Rius R. (1997). La greffe de fichiers et ses conditions d'application. Méthode et exemple. In: Brossier G., Dussaix A.M. (Eds), *Enquêtes et sondages.* Dunod, Paris, 94-102.

[3] Aluja-Banet T., Rius R., Nonell R., Martínez-Abarca M.J. (1998) Data Fusion and File Grafting. *Analyses Multidimensionelles Des Données.* NGUS 97. 1 ed. París: CISIA-CERESTA, Eds. A. Morineau, K. Fernández Aguirre, P. 7-14.

[4] Aluja-Banet T., Daunis-i-Estadella J., Pellicer D. (2007). GRAFT, a complete system fro data fusion. *Computational statistics and data analysis* 52, 635 - 649.

[5] Aria M., D'Ambrosio A., Siciliano R. (2007) Robust Incremental Trees for Missing Data Imputation and Data Fusion. *Classification and Data Analisys 2007, Book of short papers (Macerata, September 12-14, 2007)*, EUM Macerata, 287-290.

[6] Aria, M., Siciliano, R. (2003). Learning from Trees: Two-Stage Enhancements. *In Proceedings of Classification and Data Analysis Group (CLADAG 2003)*, 22-24 Settembre, Bologna.

[7] Barcena, M.J., Tusell, F. (1999). Enlace de encuestas: una propuesta metodológica y aplicación a la Encuesta de Presupuestos de Tempo. *Qüestiio*, vol. 23, núm. 2, pp. 297–320.

[8] Batista G., Monard M. C. (2003). An Analysis of Four Missing Data Treatment Methods for Supervised Learning. *Applied Artificial Intelligence*

[9] Benzecri, J.P. (1973). *L'Analyse des Données*, 2 Vols. Dunod, Paris, France.

[10] Bogart K.P. (1975). Preference structure II: distances between asymmetric relations. *SIAM Journal on Applied Mathematics* 29.

[11] Bolasco, S. (1997). *Analisi Multidimensionale dei Dati, Metodi, Strategie e Criteri di Interpretazione.* Carocci.

[12] Bonnefous S., Brenot J., Pagés J.P. (1986). Méthode de la greffe et communication entre enquêtes. *Data analysis and Informatics IV.*

[13] Breiman, L. (1996). Bagging Predictors, *Machine Learning*, 26, 46-59.

[14] Breiman L. (1996). Bias, Variance and Arcing Classifiers. *Dept. Of Statistics, University of California. Technical Report.*

[15] Brieman L.(1998). Arcing classifiers. *The Annals of statistics*, 26(3).

[16] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). *Classification and Regression Trees.* Wadsworth International Group, Belmont, California.

[17] Buja, A., Lee, Y.S. (1999). A data mining criteria for tree based regression and classification. Technical report, At & T Labs. www.research.att/andreas/papers/tree.ps.gz.

[18] Cherkassky V., Mulier F. (1998). *Learning from Data: concepts, theory, and methods*. John Wiley & Sons., New York, USA.

[19] Chu C.K., Cheng P.E. (1995). Nonparametric regression estimation with missing data. *Journal of Statistical Planning and Inference*, 48, 85-99.

[20] Ciampi, A. (1994). *Classification and discrimination: the RECPAM approach*, COMPSTAT'94, Dutter R. and Grossmann W. eds, Phisica-Verlag, Heidelberg, 129-147.

[21] CISIA - CERESIA (2001), SPAD version 5.0, Manuel de Prise en Main, CISIA-CESTA, Montreuil, France.

[22] Clogg, C.C., Shihadeh E.S. (1994). *Statistical Models for Ordinal Variables*, Thousand Oaks, CA.: Sage Publications.

[23] Conversano, C., Mola, F., Siciliano, R. (2001). Partitioning and Combined Model Integration for Data Mining, presented at the Symposium on Data Mining and Statistics (Augsburg, November 2000), *Journal of Computational Statistics*, 16, 323-339, Physica Verlag, Heidelberg (D).

[24] Conversano, C., Mola, F., Siciliano, R. (2000). Generalized Additive Multi-Model for Classification and Prediction, in H.A.L. Kiers, J.P. Rasson, P.J.F. Groen, M. Shader (Eds.): *Data Analysis, Classification and Related Methods*, Springer Verlag, Berlin (D), 205-210.

[25] Conversano, C., Siciliano, R., Mola, F., (2000). Supervised Classifier Combination through Generalized Additive Multi-Model, in

F. Roli, J. Kittler (Eds.): *Proceedings of the First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, Physica Verlag, Heidelberg (D), 167-176.

[26] Cover, T., Thomas, J. (1991). *Elements of Information Theory.* Wiley, New York.

[27] D'Ambra L., Lauro N.C. (1982). Analisi in componenti principali in rapporto ad un sottospazio di riferimento. *Rivista di Statistica Applicata*, 15, 1-25.

[28] D'Ambrosio A., Aria M., Siciliano R. (2007). Robust Tree-based Incremental Imputation Method for Data Fusion. *Advances in Intelligent Data Analysis*, Springer-Verlag, pp 174-183.

[29] De'ath G. (2002). Multivariate regression trees: a new technique for modeling species-environment relationships. *Echology* 83

[30] de Leeuw, J., van der Heijden, P.G.M. (1991). Reduced-rank models for contingency tables, *Biometrika*, 78, 229-232.

[31] DesJardins D. (1997). Coursebook for exploratory data analysis and graphics. Statistical Research Division, U.S. Bureau of the Census, Washington DC, 202333-91100, USA.

[32] Dietterich T.G. (2000). Ensemble methods in machine learning. In J.Kittler and F.Roli, editors, multiple classifier system. *First International Workshop, MCS 2000,Cagliari*, volume 1857 of lecture notes in computer science. Springer-Verlag.

[33] Dietterich T.G., Bakiri G. (1995) Solving multiclass learning problems via error-connecting output codes. *Journal of Artificial Intelligence Research*, 2.

[34] Ditrich R., Hatzinger R., Katzenbeisser W. (1998). Modelling the effect of subject-specific covariates in paired comparison studies with an application to university rankings. *Applied Statistics* 47.

[35] Ditrich R., Katzenbeisser W, Hatzinger R.,. (2000). The analysis of rank order preference data based on Bradley-Terry Type models. *OR Spectrum* 22.

[36] Drucker H. (1997). Improving regressors using boosting techniques. *In proceedings of the 14th International Conference on Machine Learning*, pages 107-115. Morgan Kaufmann.

[37] Duda, R., Hart, P., Stork, D. (2000). *Pattern Classification (Second Edition)*. Wiley, New York.

[38] Eibl, G., Pfeiffer, K. P. (2002). How to make AdaBoost.M1 work for weak base classifiers by changing only one line of the code. *Machine Learning: ECML 2002, Lecture Notes in Artificial Intelligence*. Springer.

[39] Efron, B., (1979). Bootstrap methods: Another look at the Jackknife, *Annals of Statistics*, 7, pp. 1-26.

[40] Efron, B., Tibshirani, R.J. (1993). *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability 57. London: Chapman and Hall.

[41] Efron, B., Tibshirani, R.J. (1993). Statistical analysis in the computer age, *Science*, 253: 390-395.

[42] Emond E.J., Mason D.W. (2002). A new rank correlation coefficient with application to the consensus ranking problem. *Journal of Multi-Criteria Decision Analysis.*

[43] Fabbris, L. (1997). *Statistica Multivariata.* McGraw-Hill.

[44] Feigin P.D., Cohen A. (1978). On a model for concordance between judges. *Journal of the Royal Statistical Society*, B, 40, 203-213.

[45] Fellegi I. P., and Holt D. (1976). A systematic approach to automatic edit and imputation. *Journal of American Statistical Association*, 71, 17-35.

[46] Freund Y. (1995) Boosting a weak learning algorithm by majority. *Information e computation.*

[47] Freund Y., Schapire R.E. (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1).

[48] Friedman, J.H.F., (1994). An overview of predictive learning and function approximation, in V.Cherkassy, J.Friedman, H.Wechsler (eds), *From Statistics to Neural Networks*, Vol.136 of NATO ISI Series F, Springer Verlag, New York.

[49] Friedman J.H. (1997) On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery* 1.

[50] Friedman, J.H.F., Hastie, T., Tisbshirani, R. (2000). Additive logistic regression: a statistical view of boosting, *Annal of Statistics*, 28, 377-386.

[51] Friedman, J.H., Popescu, B.E. (2005). Predictive Learning via Rule Ensembles, *Technical Report of Stanford University.*

[52] Friedman J.H., Hall P. (1999). On Bagging and Nonlinear Estimation. *Tecnical report*, University of Stanford, ttp://www-stat.stanford.edu/ jhf/#Reports.

[53] Fligner M.A., Verducci J.S. (1988) . Multistage rankings models. *Journal of the American Statistical Association* 83.

[54] Francis B., Ditrich R., Hatzinger R., Penn R. (2002). Analysing partial ranks by using smoothed paired comparison methods: an investigation of value orientation in Europe. *Applied Statistics* 51.

[55] Geman S., Doursat E.B.R. (1992) Neural Networks and the Bias/Variance dilemma. *Neural Computation*,4.

[56] Gey, S., Poggi, J.M. (2006). Boosting and instability for regression trees. *Computational Statistics and Data Analysis*, 50, 533-550.

[57] Goodman, L.A., Kruskal, W.H. (1954). Measures of association for cross-classification. *Journal of American Statistical Association*, 48, 732-762.

[58] Gordon, A. (1999). *Classification (Second Edition)*, Chapman and Hall/CRC Press, London.

[59] Hand, D., (1998). Data Mining,: Statistics or more?. *Am.Statist.*, 52, 112-118.

[60] Hand.D., Mannila H., Smyth P. (2001). *Principles of Data Mining.* A Bradford Book, The MIT Press, Cambridge, Massachusetts, London, England.

[61] Hastie, T.J., Tibshirani, R.J., Friedman, J.H. (2001). *The Elements of Statistical Learning.* Springer Verlag.

[62] Heiser W.J. (2004). Geometric representation of association between categories. *Psychometrica.*

[63] Hsiao W.C., Shih Y.S. (2007). Splitting variable selection for multivariate regression trees. *Statistics & Probability Letters* 77, pag. 265-271.

[64] Jobson, J.D.,(1992). *Applied Multivariate Data Analysis Volume I: Regression and Experimental Design.* Springer-Verlag, New York.

[65] Jobson, J.D.,(1992). *Applied Multivariate Data Analysis Volume II: Categorical and Multivariate Methods.* Springer-Verlag, New York.

[66] Kim, H., Loh, W.Y. (2001). Classification Trees with Unbiased Multiway Splits, *Journal of the American Statistical Association*, 96, 454, 589-604.

[67] Kemeny J. G. (1962). *Mathematical Models in the Social Sciences.* Ginn and Company.

[68] Kohavi R., Wolpert D.H. (1996) Bias plus Variance Decomposition for Zero-One Loss Functions. *Machine Learning: Proceedings of the Thirteenth International Conference.*

[69] Ibrahim, J.G., Lipsitz, S.R., Chen, M.H. (1999). Missing Covariates in Generalized Linear Models when the missing data mechanism is nonignorable. *Journal of the Royal Statistical Society*, Series B, 61(1). 173-190.

[70] Larsen D.R., Speckman C.L. (2004). Multivariate regression trees for analysis of abundance data. *Biometrics*, 60. 543-459.

[71] Lauro, N.C., Siciliano, R. (1989). Exploratory methods and modelling for contigency tables analisys: an integrated approach. *Statistica Applicata*, 1.

[72] Little J.R.A, Rubin, D.B. (1987). *Statistical Analysis with Missing Data.* John Wiley and Sons, New York.

[73] Little J.R.A. (1992). Regression with missing X's: A review. *Journal of the American Statistical Association*, 87(420), 1227-1237.

[74] Litvak B.G. (1983). Distances and consensus rankings. *Cybernetics and System Analysis.*

[75] Loh, W., Vanichsetakul, N. (1988). Tree-Structured Classification via Generalized Discriminant Analysis. *Journal of the American Statistical Association*, 83, 715-728.

[76] Lubinsky, D. J. (1995). Increasing the performance and consistency of classification trees by using the accuracy criterion at the leaves. *In Proceedings of the Twelth International Conference on Machine Learning*, 371-377 Taho City, Ca. Morgan Kaufmann.

[77] Maclin R., Optiz D. (1997). An empirical evaluation of Bagging and Boosting. *Fourteenth National Conference on Artificial Intelligence.* Providence, Rhode Island.

[78] Marden J.I. (1995). *Analyzing and modelling rank data.* Chapman & Hall, London.

[79] Martinez W.L., Martinez, A.R., (2002). *Computational Statistics Handbook with MatLab.* Chapman & Hall/CRC, Boca Raton, Florida.

[80] Meir R., Ratsch G. (2003). An introduction to Boosting and Leveraging. In S. Mendelson and A. Smola, editors, *Advanced Lectures on Machine Learning*, LNCS. Springer, 2003.

[81] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). *Equations of state calculations by fast computing machines.* J Chem Phys, 21:1087-1091.

[82] Mola, F. (1993). *Aspetti metodologici e computazionali delle tecniche di segmentazione binaria: Un contributo basato su una funzione di predizione.* Unpublished Ph.D. thesis, Dipartimento di Matematica e Statistica, Universitá degli Studi di Napoli Federico II.

[83] Mola, F., Siciliano, R. (1998). A general splitting criterion for classification trees, *Metron*, 56, 3-4.

[84] Mola, F., Siciliano, R. (1997). A Fast Splitting Procedure for Classification and Regression Trees, *Statistics and Computing*, 7, Chapman Hall, 208-216.

[85] Mola, F., Siciliano, R. (1994). Alternative strategies and CATANOVA testing in two-stage binary segmentation, in E. Diday, Y. Lechevallier, M. Schader, P. Bertrand, B. Burtschy (Eds.): *New Approaches in Classification and Data Analysis: Proceedings of IFCS 93*, Springer Verlag, Heidelberg (D), 316-323.

[86] Mola, F., Siciliano, R. (1992). A two-stage predictive splitting algorithm in binary segmentation, in Y. Dodge, J. Whittaker. (Eds.): *Computational Statistics: COMPSTAT 92*, 1, Physica Verlag, Heidelberg (D), 179-184.

[87] Murphy, P. M. and Aha, D. W. (1993). UCI repository of machine learning databases. *Machine-readable data repository.* University of California, Department of Information and Computer Science, Irvine, CA.

[88] Petrakos, G., Conversano, C., Farmakis, G., Mola, F., Siciliano, R., Stavropoulos, P. (2004) New ways to specify data edits. *Journal of Royal Statistical Society*, Series A, volume 167, part 2, 249-274.

[89] Piccolo D. (2000) *Statistica.* Il Mulino.

[90] Piscitelli A. (2005). *Data fusion: un approccio non simmetrico al file grafting.* Unpublished Ph.D. thesis, Dipartimento di Matematica e Statistica, Universitá degli Studi di Napoli Federico II.

[91] Quinlan, J. R. (1993). *C4.5: Programs For Machine Learning.* Morgan Kaufmann, Los Altos.

[92] Rizzi, A. (1985). *Analisi dei Dati.* La Nuova Italia Scientifica.

[93] Rius R., Nonell R., Aluja-Banet T. (1996). File grafting: A data sets communication tool. *In proceedings in Computational Statistics* COMPSTAT 9́6.

[94] Rius R., Aluja-Banet T., Nonell R. (1999). File grafting in market research. *Applied Stochastic Models in Business and Industry* 15, 451 - 460

[95] Rubin, D.B. (2003). Discussion on multiple imputation. *Internantiona statistical review* 71, 3, pp. 619-625.

[96] Rubin, D.B. (1976). Inference and Missing Data (with Discussion). *Biometrika* 63, pp.581-592.

[97] Saporta, G. (2002) Data fusion and data grafting. *Computational Statistics and Data Analysis* 38, 465-473.

[98] Sarle, W.S. (1998). *Prediction with Missing Inputs* Technical Report, SAS Institute.

[99] Schafer, J. L., (1997). *Analysis of Incomplete Multivariate Data.* Chapman & Hall.

[100] Schapire R. E. (1999). A brief introduction to Boosting. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence.*

[101] Schapire R.E. (1990) The strength of weak learnability. *Machine learning* 5(2).

[102] Schapire R.E., Freund Y., Barlett P., Lee W.S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5).

[103] Schapire R.E., Singer Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning* 37(3).

[104] Shannon W.D., Banks, D. (1999). Combining classification trees using mle. *Statistical in Medicine*, 18:727-740.

[105] Siciliano R., Mola F. (2000). Multivariate data analysis and modelling through classification and regression trees. *Computational Statistics & Data Analysis.*

[106] Siciliano R., Aria., D'Ambrosio A. (2006), Boosted incremental tree-based imputation of missing data. *Data Analysis, Classification and the Forward Search. Springer series in Studies in Classification, Data Analysis, and Knowledge Organization.* Springer-Verlag, pp. 271-278.

[107] Siciliano R., Aria., D'Ambrosio A. (2005), Boosted stump algorithm for missing data incremental imputation. *CLADAG 2005, Book of Short Papers (Parma, June 6-8, 2005)*, MUP, Parma, 161-164.

[108] Siciliano, R. (1999). Latent budget trees for multiple classification, in M. Vichi, P. Optitz (Eds.): *Classification and Data*

Analysis: Theory and Application, Springer Verlag, Heidelberg (D).

[109] Siciliano, R. (1998). Exploratory versus Decision Trees, invited lecture to COMPSTAT '98 (Bristol, August 24-28), in R. Payne, P. Green (Eds.): *Proceedings in Computational statistics: 13th Symposium of COMPSTAT*, Physica Verlag, Heidelberg (D).

[110] Siciliano, R., Mola, F. (2000). Multivariate Data Analysis through Classification and Regression Trees, *Computational Statistics and Data Analysis*, 32, 285-301, Elsevier Science.

[111] Siciliano, R., Mola, F. (2002). Discriminant Analysis and Factorial Multiple Splits in Recursive Partitioning for Data Mining, in Roli, F., Kittler, J. (eds.): *Proceedings of International Conference on Multiple Classifier Systems* (Chia, June 24-26, 2002), 118-126, Lecture Notes in Computer Science, Springer, Heidelberg.

[112] Siciliano R., Conversano C. (2002). Tree-based Classifiers for Conditional Missing Data Incremental Imputation. *Proceedings of the International Conference on Data Clean* (Jyväskylä, May 29-31, 2002), University of Jyväskylä.

[113] Siciliano, R., Mooijaart, A. (1999). Unconditional Latent Budget Analysis: a Neural Network Approach, in S. Borra, R. Rocci, M. Vichi, M. Schader (Eds.): *Advances in Classification and Data Analysis*, Springer-Verlag, Berlin, 127-136.

[114] Steverink M.H.M., Heiser W.J., van der Kloot W.A. (2002). Avoiding degenerate solutions in multidimensional unfolding by using additional distance information. *Technical report of University of Leiden.*

[115] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions, *Journal of the Rojal Statistical Society*, Series B, Vol. 36, pp. 111-133.

[116] Takeuchi, K., Yanai, H., Mukherjee, B. (1982). *The Foundations of Multivariate Analysis*, Wiley Eastern, New Dehli.

[117] Thisted, R.A., (1988). *Elements of Statistical Computing: Numerical Computation.* London, Chapman and Hall.

[118] Tibshirani R. (1996). Bias, variance and prediction error for classification rules. *Technical report*, University of Toronto.

[119] Urbanek, S. (2002). Different ways to see a tree - KLIMT, in *Proc. of the 14th Conference on Computational Statistics*, (Compstat 2002), p303-308, Physica, Heidelberg.

[120] Vach, W. (1994). Logistic Regression with Missing Values and Covariates, *Lecture Notes in Statistics*, vol. 86, Springer Verlag, Berlin.

[121] Valiant L.G. (1999). A theory of the learnable. *Communication of the ACM* 27/11.

[122] van der Ark, L.A. (1999). *Contributions to Latent Budget Analysis. A Tool for the Analysis of Compositional Data.* DSWO Press, Leiden University.

[123] van Brokland-Vogelesang R. (1990). Unfolding and group consensus anking for individual preferences. *PhD thesis, University of Leiden.*

[124] van der Putten, P. (2000). Data Fusion for Data Mining: a Problem Statement. *Coil Seminar 2000*, Chios, Greece, June 22-23.

[125] van der Putten, P., Kok, J.N., Gupta, A. (2002). Data Fusion through Statistical Matching. *MIT Sloan School of Management Working* Paper No. 4342-02, Cambridge, MA.

[126] Vapnik, V. (1995). *The Nature of Statistical Learning Theory.* Springer-Verlag, Berlin.

[127] Vapnik, V. (1998). *Statistical Learning Theory.* Chichester, John Wiley & Sons, United Kingdom.

[128] Winkler W. E. (1999). State of statistical data editing and current research problems. *Working paper No 29 in the UN/ECE Work Session on Statistical Data Editing*, Rome, 2-4 June 1999.

[129] Wolpert D. (1997). On bias plus variance. *Neural Computation* 9[6], pp. 1211 - 1243.

[130] Zani, S. (2000) *Analisi dei dati statistici, vol. II, Osservazioni multidimensionali*, Giuffré ed., Milano.

[131] Zani, S. (1998) *Analisi dei dati statistici, vol. I, Osservazioni in una e due dimensioni*, Giuffré ed., Milano.