VSB — TECHNICAL UNIVERSITY OF OSTRAVA

FACULTY OF ECONOMICS

DEPARTMENT OF FINANCE

Assessing the Creditworthiness of a Client Using Classification Trees

Posuzování bonity klienta pomocí klasifikačních stromů

Student: Songyang Gao

Supervisor of the bachelor thesis: Mgr. Taťána Funioková, Ph.D.

Ostrava, 2019

VŠB - Technical University of Ostrava
Faculty of Economics
Department of Finance

Bachelor Thesis Assignment

Student:	**Songyang Gao**
Study Programme:	B6202 Economic Policy and Administration
Study Branch:	6202R010 Finance
Title:	Assessing the Creditworthiness of a Client Using Classification Trees Posuzování bonity klienta pomocí klasifikačních stromů
The thesis language:	English

Description:

1. Introduction
2. Description of the CART Decision Tree Methodology
3. Implementation of the CART Algorithm in R
4. Application for Classification and Prediction
5. Conclusion

Bibliography
List of Abbreviations
Declaration of Utilisation of Results from the Bachelor Thesis
List of Annexes
Annexes

References:

BREIMAN, L., J. H. FRIEDMAN, R. A. OLSHEN and C. J. STONE. *Classification and regression trees.* New York: Chapman & Hall, 1993. ISBN 978-0-412-04841-8.
DALGAARD, Peter. *Introductory statistics with R.* New York: Springer, c2002. ISBN 0-387-95475-9.
HAND, J. David and William, E. HENLEY. Statistical Classification Methods in Consumer Credit Scoring: a Review. *Journal of the Royal Statistical Society Series A.* 1997, Vol. 160, Issue 3, p. 523. ISSN 0964-1998.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Mgr. Taťána Funioková, Ph.D.**

Date of issue:	23.11.2018
Date of submission:	10.05.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA OSTRAVA
Ekonomická fakulta
1

[illegible]
Ing. Iveta Ratmanová, Ph.D.
Head of Department

[illegible]
prof. Dr. Ing. Zdeněk Zmeškal
Dean

The declaration

“I hereby declare that I have elaborated the entire thesis including annexes myself.”

Ostrava dated……… 10. 5. 2019

Songyang Gao 高淞扬

……………………………………

SONGYANG GAO

CONTENTS

1 Introduction

Credit scoring, also known as credit rating and credit evaluation, is a social intermediary service that provides credit information to the society or provides decision-making reference for the unit itself. Originally produced in the United States in the early 20th century.

The credit-granting process leads to a choice between two actions—give this new applicant credit or refuse this applicant credit. Credit scoring tries to assist this decision by finding what would have been the better rule to apply on a sample of previous applicants. The advantage of doing this is that we know how these applicants subsequently performed. We can improve the found rule and apply it to new applicant.

The structure of thesis is the following.

Chapter 1 is a brief introduction to the thesis.

Chapter 2 is the description of the CART decision tree methodology. In this chapter, we briefly describe also other methods. Then CART method is the only one we use to classify and predict the data. Then it is the notation of the CART we use to explain the algorithm.

Chapter 3 is the implementation of the CART algorithm in R. We just briefly describe programming language R and R studio. This chapter includes a list of the notation about the packages and function we will use in the application part to build an optimal model to describe it and use for the prediction.

Chapter 4 shows the application for classification and prediction. In this chapter, we simply explain the credit scoring and describe the data we used to analyze. Then in the growing phase, we describe the way of splitting for the first two splits in detail and describe how to build the maximal tree. As for the accuracy of the prediction, we show how the classes can be predicted and how the confusion matrices can be generated. With pruning phase, we can get the optimal tree within expectations. We introduce penalty to control the bias-variance trade-off and to apply structural risk minimization. Finally, we evaluate the performance of the model with the testing data.

Chapter 5 is the conclusion. It shows our prediction result.

2 Description of the CART Decision Tree methodology

Classification and regression trees (CART) are machine-learning methods for constructing prediction models from data. The models are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition. As a result, the partitioning can be represented graphically as a decision tree. Classification trees are designed for dependent variables that take a finite number of unordered or ordered values, with prediction error measured in terms of misclassification cost. Regression trees are for dependent variables that take continues values, with prediction error typically measured by the squared difference between the observed and predicted values. Machine learning is the process of automatically discovering patterns and trends in data that go beyond simple analysis. There is a great deal of overlap between learning algorithms and statistics and most of the techniques used in learning algorithms can be placed in a statistical framework. Statistical models usually make strong assumptions about the data and, based on those assumptions, they make strong statements about the results.

2.1 Overview of alternative regression and classification methods

We will introduce only five alternative methods in the subchapter. Three of them based upon analysis of variance: linear regression, logistic regression and diskriminant anaysis and two other methods: one rule method and Bayesian method, based upon frequency tables similarly as the recursive partitioning is.

2.1.1 Regression

Regression analysis is a statistical analysis method that determines the quantitative relationship between two or more variables. The application is very extensive. It is one of the most known modeling techniques and often one of the preferred techniques for people to learn predictive models. In this technique, the dependent variable is continuous, the independent variable can be continuous or discrete.

The regression analysis is divided into one-way (simple) regression and multiple regression analysis according to the variables involved. According to expected relationship between independent variables and dependent variables it can be divided into linear regression analysis and nonlinear regression analysis. If in the regression analysis, only one independent variable and one dependent variable are included, and the relationship between the two can be approximated by a straight line, this regression analysis is called a simple linear regression analysis. If two or more independent variables are included in the regression analysis and there is a linear correlation between the independent variables, it is called multiple linear regression analysis.

The multiple linear regression (MLR) serves to predict one continuous dependent (criterion) variable Y with use of one or more independent explanatory (predictor) variables, $X_1 ... X_n$. It will estimate a linear

equation of the form:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$$

where β_0 ,..., β_n are unknown parameters and values of Y are then predicted by

$$y_i = \beta_0 + \beta_1 X_{i1} + \ldots + \beta_n X_{in} + \varepsilon_i ,$$

where ε it the model's error term (also known as the residuals).

X_i can be continuous or categorical. For categorical variables we have to create numerical dummy variables to represent every category of the explanatory variable (it's coded with 1 if the case falls in the category and with 0 if not.) Ex post MLR analysis makes number of assumptions: linear relationship between dependent

and explanatory variables, residuals should be normally distributed with mean= 0, independent variables are not highly correlated with each other. The flaw of this method is, that it assumes a continuous dependent variable Y, so I could take any value from -∞ to +∞. This is the reason why to prefer logistic regression for classification problem.

2.1.2 Logistic regression

Logistic regression (or logit regression) is a generalized linear model and therefore has many similarities with MLR. The dependent variable of logistic regression can be two-category or multi-category, but the two-category is more commonly used and easier to explain. The most commonly used in practice is the logistic regression of the two classifications, the so-called binary logistic regression (BLR). It means the type of the dependent variable belongs to the binary (1 / 0, true / false, yes / nom, good/bad) variable and we use logistic regression to calculate the probability of "event=Success" and "event=Failure". Their model forms are basically the same, both have $\beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$, the difference is that their dependent variables are different, directly uses $\beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$ as the dependent variable, $Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$ and logistic regression assigns $\beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$to a hidden state p which always falls in the range of 0 and 1. Suppose we think of the binary target variable Y in term of an underlying probability P, then the model is

$$P = e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_{n1}}$$

Then we determine the dependent variable according to the size of p and 1-p. value via the logit transformation:

$$\ln \frac{p}{1-p} = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$$

where,

$$P \text{ or } (1 - P)$$

is an odds ratio, and

$$\ln \frac{p}{1-p}$$

is the log odds (log of the odds ratio).

Compared to MLR, logistic regression does not apply so many assumptions. For BLR, the dependent variable has to be binary, little or no collinearity among the independent variables, observations should not come from repeated measurements or matched data (observations are independent of each other), linearity of independent variables and log odds, and a large sample size is required.

2.1.3 Discriminant analysis

Discriminant analysis was developed in the 1930s as a statistical method for discriminating models of unknown categories using samples of known categories. In recent years, discriminant analysis has been widely used in the disciplines of natural sciences, sociology, and economic management. The distinguishing analysis is characterized by summarizing the regularity of the classification of objective objects based on the data information of several samples of each category that has been mastered and historically, and establishing the discriminant formula and discriminant criterion. When a new sample point is encountered, the category to which the sample point belongs can be discriminated based on the discriminant formula and the criterion. The discriminant analysis is distinguished according to the number of discriminating groups, and can be divided into two groups of discriminant analysis and multi-group discriminant analysis.

In the binary classification case the discriminant functional analysis is referred to as Fisher linear discriminant analysis (LDA). The aim is to find the linear combination of predictor variables to separate two groups,

e.g. bad and good clients:

$$Y = W_1 \cdot X_1 + W_2 \cdot X_2 + \ldots + W_n \cdot X_n,$$

Where W_i are unknown weights. A measure of separation is how different are the mean values of Y for those two different groups, E (Y |"event=Success") and E (Y |"event=Failure"). We want the difference

E (Y |"event=Success")− (Y |"event=Failure") to be maximal. The LDA analysis

is identical to MLR analysis for this simple case and so we require similar assumptions to be met.

2.1.4 One Rule Method

The One rule, or 1R method, is a very simple rule-based classification algorithm that generates only one rule for each predictor in the data. It assumes discrete attributes ($X_1 \ldots X_n$). If we have continuous variable, then we can use a binning to transform it to a discrete variable. For every attribute a modal class is determined along with its relative frequency. The aim is to choose the attribute which perform the highest relative frequency of the modal class. In other words, the “One Rule” is the rule with the smallest total error. At the end we classify a new object on the rule basis of a single attribute.

2.1.5 Bayesian Method

The Naïve Bayes classifier technique is based on the well-known Bayes rule:

$$\mathrm{P(B|A)} = \mathrm{P(A|B)} \cdot \frac{\mathrm{P(B)}}{\mathrm{P(A)}}$$

Despite the fact it is a simple method, it can outperform more sophisticated classification methods. Given a set of independent variables$x_1, \ldots, x_n$continuous or categorical and a binary dependent variable Y (Success/Failure), we want to construct the posterior probability for "event=Success” and for "event=Failure”:

$$P(Success|x_1, \ldots, x_n) = P(x_1, \ldots, x_n|Successs) \cdot P(Successs),$$

and

$$P(Failure|x_1, \ldots, x_n) = P(x_1, \ldots, x_n|Failure) \cdot P(Failure).$$

Since this approach assumes that the conditional probabilities of the independent variables are statistically independent we can rewrite the first formula as

$$P(Success|x_1, \ldots, x_n) = P(Successs) \prod_{i=1}^{n} P(x_i|Successs)$$

We classify a new object with a class (Success/Failure) that achieves the highest posterior probability.

2.2 Recursive partitioning

Recursive partitioning, or CART is a statistical method for multivariate analysis. Recursive partitioning creates a decision tree that strives to correctly classify group members by dividing them into subgroups based on several binary independent variables. This process is called a recursive process because each subgroup can be split indefinitely until the segmentation process terminates after reaching a certain stopping criterion.

The aim of CART is to build a classification or regression tree for predicting continuous dependent variable (regression) or categorical predictor variable (classification). In general, the purpose is to determine a set of if-then rules that enable the most accurate prediction of classification of cases. There are numerous methods which can be used for classification problems. The process of building a classification tree can be characterized as some basic steps.

- ➢ Firstly, specifying the criteria for predictive accuracy. The most accurate prediction is defined as the prediction with the minimum costs which can be measured in terms of proportion of misclassified cases. Sometimes more accurate classification is desired for some classes than others. There are numerous approaches how to include some kind of penalty in the incorrect classification.
- ➢ The second step is to select the splits on the predictor variables that generate the highest improvement in predictive accuracy. This is usually measured with some type of node impurity measure representing level of homogeneity of cases in nodes. It can be defined, for example by using Gini index, information gain, or χ^2.
- ➢ Let the tree grow and decide when to stop splitting. In principal, splitting could continue until all cases are perfectly classified or predicted. However, we do not want end up with a tree structure that is too complex. Some algorithms apply a reasonable stopping rule. One way to control splitting is to

specify minimum number of cases or objects in terminal nodes. Another way is to allow splitting to continue until all terminal nodes are pure or contain no more than a specified minimum fraction of cases in one or more classes.

- The last step is to prune the generated tree and select the “right-sized” tree. I will describe this step in more detail in connection with the algorithm I use in the next Chapter.

2.2.1 If-then rules and conditional probability distribution

We can think of a decision tree as a collection of if-then rules. The process of converting a decision tree into an if-then rule is like: constructing a rule from the root node of the decision tree to each path of the leaf node. The characteristics of the internal nodes correspond to the conditions of the rules, and the classes of the leaf nodes correspond to the conclusion of the rules. The path of the decision tree or its corresponding if-then rule set has an important property: mutual exclusive and completeness. Each instance is covered by a path or a rule, and is only covered by a path or a rule. It refers to the condition that the feature of the instance is consistent with the feature on the path or the instance satisfies the rule.

The decision tree essentially abstracts a set of classification rules from the training data set. What I need is a decision tree with less contradiction to the training data, and it has good generalization ability. From another perspective, the decision tree is a conditional probability model estimated by the training data set. There are infinite number of conditional probability models for classes based on feature space partitioning. The conditional probability model I choose should not only have a good fit to the training data, but also have a good prediction of the unknown data.

The decision tree has the loss function to represent this goal. As described below, the loss function of the decision tree is a regularized maximum likelihood function, and the decision tree strategy is to minimize the loss function as the objective function. The decision tree learning algorithm in reality adopts a heuristic method, approximation. Solve this optimization problem, and the resulting decision tree is sub-optimal.

2.2.2 CART algorithm and notation

The classification and regression tree (CART) model was proposed by Breiman et al. in 1984 and is a widely used decision tree learning method. CART is also composed of feature selection, tree generation and pruning, which can be used for classification.

The tree assumes that the decision tree is a binary tree, the values of the internal node features are "yes" and "no".

The decision tree algorithm is a recursive selection of the optimal feature, and the training data is segmented according to the feature, so that there is a best classification process for each sub-data set. This process corresponds to the division of the feature space. Corresponding to the construction and start, the root node is constructed, all training data is placed at the root node, and an optimal feature is selected. According to this feature, the training data set is divided into subsets, so that each subset has the best classification under current conditions. If these subsets can already be classified correctly, then construct the leaf nodes and assign them to the corresponding leaf nodes; if there are still subsets that cannot be of basically correct classification, then select new optimal features for these subsets, continue to segment them, build corresponding nodes, and proceed recursively until all training data subsets are correctly classified or have no suitable features. Finally, each subset is assigned to the leaf node, that is, there is a clear class. This generates a decision tree.

The decision tree generated by the above method may have a good classification ability for the training data, but – for an unknown test data it may not have a good classification ability, that is, the fitting phenomenon may occur. I need to generate candidates for optimal tree from the bottom. Pruning on the tree makes it simpler, so that it has better generalization ability. Specifically, it removes the sub-segmented leaf nodes and makes them fall back to the parent node, even higher node. That is, then change the parent node or higher node to a new leaf node.

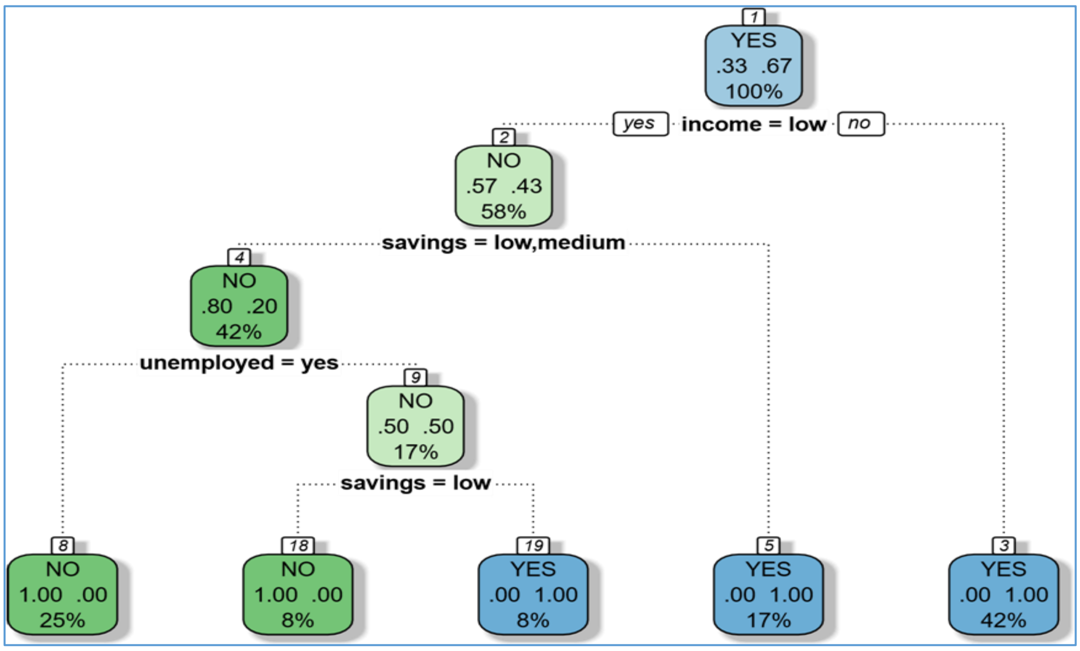
The decision tree is a basic classification and regression method. The decision tree model is a tree structure. In the classification problem, it represents the process of classifying instances based on independent values. It can also be considered as a conditional probability distribution defined in feature space and class space. Its main advantage is that the model is clear, and the classification speed is fast. When learning, training data, the decision tree model is established according to the principle of minimizing the loss function. In the prediction, the new data is classified by the decision tree model. The ideas of these decision tree are mainly derived from the CART algorithm proposed by Breiman et al (1984), the ID3 algorithm proposed by Quinlan (1986) and the C4.5 algorithm (1993).

The classification decision tree model structure that describes the classification of instances. The decision tree consists of nodes and directed edges. There are two types of nodes: internal nodes and leaf nodes. The inner node represents a feature or attribute, and the leaf node represents a class. Generally, an internal node (also known as an inner node or branch node) is any node of a tree that has child nodes. Similarly, a leaf node (also known as an outer node or terminal node) is any node that does not have child nodes. The height of a tree is the length of the longest path to a leaf and the depth of a node is the length of the path to its root.

The Fig 2.1 is a schematic diagram of a decision tree with 5 leaf nodes [marked with 8,18,19,5,3] and4 internal nodes [marked with1,2,4,9]. Number of leaf nodes corresponds to number of splits in the tree. I can see four (five minus one) splits in the figure. The height is 4 and depth of node marked with 8 is 3.

The number 33 means that 33% of learning data are “Yes” class. The number 67 means that 67% of these learning data are “No” class. And sum of percentages in node 2 and 3, namely 58% (income is low) and 42% (income is not low) is equal to 100% which is in node 1.

Figure2.1 Classification tree example generated with rattle package in R Studio

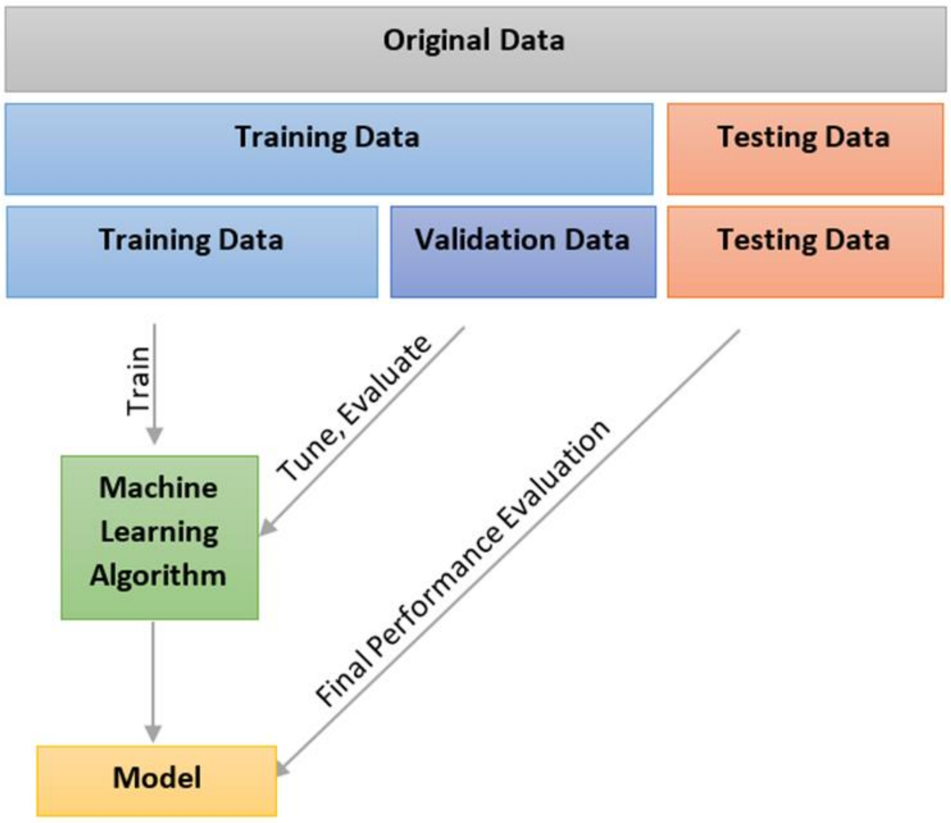


The CART algorithm consists of the following two steps:

The decision tree is generated based on the training data set, and the generated decision tree should be as large as possible.

In the Fig 2.2. We divide our original data into two groups, training data set and testing data set. And split the training data to two parts named training data and validation data. The validation data set is for evaluation. In the end, we evaluate the performance of the model with the testing data.

Figure 2.2 Data split



Source: https://medium.freecodecamp.org/how-to-get-a-grip-on-crossvalidations-bb0ba779e21c

Pruning the generated tree and selecting the best subtree by cross-validation. At mean time, it reduces the size of decision trees by removing sections of the tree that provide little hard to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

In this subsection I introduce the basic notation and terminology which will be easy to describe the CART algorithm in detail.

Assume I have n predictor variables $X_1, X_2, \dots, X_n$, these can be both categorical and numerical. Suppose the cases fall into J classes:$C_1, \dots, C_J$. The construction of a classifier is based on a learning sample of N cases. Every case x from the learning data set can be represented by a vector of predictor variables and its true class.

Learning data set (N cases):

$$(x_{11}, x_{12}, \dots, x_{1n}, j_1)$$

$$\vdots$$

$$(x_{N1}, x_{N2}, \dots, x_{Nn}, j_N)$$

Let N_j be number of cases in class j, then proportion N_j/N, $j = 1, \dots, C$ is a relative frequency of class C_j. A population proportion of classC_j, referred to as the prior class probability, is denoted by π_j.

For given a classifier d—classification rule represented by a classification tree—I obtain structure of subsets represented by nodes. Let t be a node, then $d(t)$ is a class assigned to node t by the classifier$d$, N_t is number of observations in node t, and $N_j(t)$ the number of class j cases in t.

Probability of node t for future observations is then estimated by the formula

$$P(t) = \sum_{j=1}^{C} \pi_j \frac{N_j(t)}{N_j}$$

Similarly, given node t, the probability of class j in the node t is estimated by

$$P(j|t) = \pi_j \frac{N_j(t)}{N_j} \sum_{i=1}^{C} \pi_i \frac{N_i(t)}{N_i}$$

When I set the prior probabilities π_j equal to observed class relative frequencies in the sample, that is, $\pi_j = \frac{N_j}{N}$

$$P(t) = \sum_{j=1}^{C} \frac{N_j}{N} \frac{N_j(t)}{N_j} = \frac{N_t}{N}$$

And

$$P(j|t) = \frac{N_j(t)}{N_t}$$

2.2.3 Decision tree generation

The decision tree generation algorithm recursively generates the decision tree until it cannot continue. The solution to this problem is to consider the complexity of the decision tree and simplify the generated decision tree.

In the classification process, assuming J classes, the goodness of split criterion is measured by an impurity function$f_{imp}(p_1, \dots p_J)$, where p_j represents the probability that the sample cases belong to class j The CART algorithm I used in the thesis.

Gini impurity function

$$f_{Gini}(p_1, \dots p_J) = \sum_{i=1}^{J} p_i \cdot (1 - p_i) = 1 - \sum_{i=1}^{J} p_i^2.$$

That is, impurity of a node t is given by

$$I(t) = \sum_{i=1}^{J} P\,(i|t) \cdot (1 - P(i|t)) = 1 - \sum_{i=1}^{J} P\,(i|t)^2.$$

For the two-class classification problem, if the probability that the sample point belongs to the first class is p, then the Gini index of the probability distribution is:

$$Gini(p) = 2p(1 - p)$$

And so the impurity of a node t simplifies to,

$$I(t) = 2 \cdot P(1|t) \cdot \big(1 - P(1|t)\big) = 2 \cdot P(1|t) \cdot P(2|t).$$

If I split node t into two child nodes t_L and t_R, I define the decrease in impurity as follows:

$$\Delta(t, t_L, t_R) = I(t) - P(t_L) \cdot I(t_L) - P(t_R) \cdot I(t_R)$$

I select the split that maximize$\Delta(t, t_L, t_R)$. In other words, I select the split that minimize $P(t_L) \cdot I(t_L) + P(t_R) \cdot I(t_R)$

According to the training data set, starting from the root node, recursively perform the following operations on each node to construct a binary decision tree (maximal if possible).

2.2.4 Decision tree pruning

The pruning of the decision tree is often achieved by minimizing the loss function or cost function of the decision tree as a whole. CART mean the so-called cost-complexity pruning.

If the cost of misclassifying a class j object as a class i object is the same I define misclassification rate (or misclassification cost) as

$$R(t) = 1 - \max_{j=1,\dots,C} P(j|t) \ .$$

In general, I can define the cost of misclassifying a class j object as a class i object in terms of $C(i|j)$ function, where

$$C(i|j)\begin{cases} \geq 0 & \mathrm{i} \neq \mathrm{j} \\ = 0 & \mathrm{i} = \mathrm{j} \end{cases} \tag{2.1}$$

When defining the estimate $R(t)$ of the expected misclassification cost, given the node t, by

$$R(t) = \min_{i} \sum_{j=1}^{C} C(i|j) P(j|t).$$

For a tree T, let $\tilde{T}$ denote a set of leaf nodes, then I define a misclassification cost of the tree T by

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

Let the number of leaf nodes in the tree be |T|. The $R(T)$ shows the prediction error of the model on the training data, that is, the degree of fitting of the model to the training data, |T| indicates the complexity of the model. Let |T| be the number of nodes in a tree T with misclassification error$R(T)$. I define the cost-complexity measure $R_\alpha(T)$ as

$$R_\alpha(T) = R(T) + \alpha|T| \tag{2.2}$$

The parameter α≥0 is so-called complexity parameter which represents penalty on the complexity of tree T measured by |T|.

Let $T(\alpha)$ be the smallest tree T for which $R_\alpha(T)$ is minimal. It can be shown that all possible values of α in $[0, +\infty)$ can be grouped into m intervals:

I_1 =[0,α_1]

I_2 =(α_1, α_2]

…

I_{m-1} =($\alpha_{m-2}, \alpha_{m-1}$]

I_m =($\alpha_{m-1}, +\infty$)

Such that for every α∈ I_i, the same $T(\alpha)$ is chosen.

At the end of the pruning phase I have obtain m trees $T(\alpha_1)$,…, $T(\alpha_m)$. If the size of the tree does not bother me at all I set α= 0. On the other hand, if I do not want the tree to be too complex, I set α= $+\infty$. Obviously, $T(0)$= $T(\alpha_1)$ is the maximal tree and $T(+\infty) = T(\alpha_m)$ is the root tree.

The pruning phase selects the model with the smallest loss function, that is, the subtree with the smallest loss function. When the value is determined, the larger the subtree, the better the fit with the training data, but the more complex the model is.

It can be seen that the decision tree generation only considers the better fitting of the training data by improving the (or). The decision tree pruning also considers reducing the model complexity by selected loss function.

2.2.5 Cross Validation

Cross Validation, sometimes called Rotation Estimation, is a practical way to statistically cut data samples into smaller subsets, which was proposed by Seymour Geisser.

In a given modeling sample, take most of the samples to build the model, leaving a small part of the sample to be forecasted with the model tree created, and the prediction error of this small part of the sample. This process continues until all samples are forecasted once and only once.

The basic idea of cross-validation is to group the original datasets in a sense, one part as a train set and the other part as a validation set or test set. First, build the training set pair. The classifier trains and then test the verification set to test the model obtained by the training as a performance indicator for evaluating the classifier.

If the given sample data is sufficient, a simple way to make model selection is to randomly set the data set.

Evaluation in the different complexity models learned, choose the model with the smallest prediction error for the verification set. Since the verification set has enough data, it is also effective to use it to select the model.

However, in many practical applications, the data is not sufficient. In order to select a good model, the cross-validation method can be used. The basic idea of cross-validation is to test the data repeatedly. The set is combined into a training set and a test set, and on this basis, training, testing, and model selection are repeated.

- Simple cross-validation

The simple cross-validation method is: first randomly divide the given data into two parts, one part as the training set and the other part as the test set (for example, 66% of the data is the training set and 33% of the data is the test set); The training set trains the model under various conditions (for example, different number of parameters) to obtain different models; evaluates the test errors of each model on the test set, and selects the model with the smallest test error.

- 10 fold cross validation

(Assume the priors π_j to be estimated from the data) I choose V--fold cross validation method, namely 10--fold cross validation to choose an optimal model. The learning data set is randomly divided into 10 subsets of the same size (as nearly as possible). Then the data is trained in 9 of 10 subsets. I use the remaining subset to test the model; repeat this process for possible choices; finally select the model with the smallest average test error in an evaluation.

At the end of the pruning phase number of models was generated, candidates for optimal model. Each of them corresponds to a level of complexity parameter α. Because any value from (α1, α2> generates the same model, $T(\alpha_2)$, also value of $\sqrt{\alpha_1 \cdot \alpha_2}$ generates $T(\alpha_2)$. I choose the following representative (typical value) for every model:

$\beta_1=0$

$\beta_2 = \sqrt{\alpha_1\alpha_2}$

…

$\beta_m = +\infty$.

I repeat the following procedure 10--times for every level of cp from $\{\beta_1 \dots \beta_m\}$:

Table 2.1. 10 fold cross-validation

Learning data set										
	Subset$_1$	Subset$_2$	Subset$_3$	Subset$_4$	Subset$_5$	Subset$_6$	Subset$_7$	Subset$_8$	Subset$_9$	Subset$_{10}$
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
	…	…	…	…	…	…	…	…	…	…
10	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST

Let subset$_i$ be a new testing data set. Using the new learning data set to construct a classifier with respect to given β, tree $T^{(i)}(\beta)$. Find the predicted class for each case in subset$_i$ and compute misclassification error for those predictions, $R(T^{(i)}(\beta))$

At the end of this process for every level of cp=β, where β is from $\{\beta_1, \dots \beta_m\}$, values $R(T^{(1)}(\beta)), \dots, R(T^{(10)}(\beta))$ are generated. Let

$$R^{CV}(\beta) = \frac{1}{10}\sum_{i=1}^{10} T^{(i)}(\beta)$$

be an estimator of true misclassification rate of tree T(β) with standard error of

$$\mathrm{SE}^{CV}(\beta) = \sqrt{\frac{\mathrm{R}^{CV}(\beta)(1-\mathrm{R}^{CV}(\beta))}{N}}.$$

Firstly, we are looking for $\beta_{\min}$ with the smallest cross-validation error R^{CV} ($\beta_{\min}$). Note that choice of $\beta_{\min}$ can be affected by the seed of the random number generator to separate learning data set into 10 subtest. Small changes can be large changes in number of terminal nodes in optimal tree. To choose the simplest tree whose accuracy is comparable to $R^{cv}(\beta_{\min})$, 1-SE rule is applied to select the right sized tree: An optimal tree, T (β_{opt}), is the tree corresponding to β_{opt} where β_{opt} is the maximum β satisfying

$$\mathrm{R}^{CV}(\beta) \leq \mathrm{R}^{CV}(\beta_{min}) + \mathrm{SE}^{CV}(\beta_{min}).$$

Finally, I consider β_{opt} to be an optimal model.

2.3 Performance evaluation

One of the techniques used to summarize the performance of a classification model is a confusion matrix. In the field of machine learning, especially statistical classification problems, confusion matrices, also known as error matrices, are a specific table layout that allows visualization of the performance of an algorithm. Usually supervised learning algorithms (in unattended learning) are often referred to as Match matrix). Each row of the matrix represents an instance in the predictive class, and each column represents an instance in the actual class (and vice versa). The name comes from the fact that it is easy to see if the system confuses two classes. It is a special contingency table with two dimensions ("actual value" and "predicted value").

Table 2.2 confusion matrix

		Actual Values		
		positive	negative	Total
Predicted	positive	TP True positive	FP False positive	TP+FP
	negative	FN Flase negative	TN True negative	FN+TN
	Total	TP+FN	FP+TN	TP+FP+FN+TN

True Positive (TP): Actual is positive, and is predicted to be positive. False Negative (FN): Actual is positive, but is predicted negative. True Negative (TN): Actual is negative, and is predicted to be negative. False Positive (FP): Actual is negative, but is predicted positive.

The accuracy of a classification is then measure in terms of ACC, defined as:

$$ACC = \frac{FP + FN}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}}.$$

3 Implementation of the CART Algorithm in R

In Chapter 4, the CART algorithm would be applied. In the thesis, we use R studio and R program language to solve given classification problem. Let me briefly introduce what is R and the packages we would use in the application. And we describe the packages what we need.

3.1 R and R Studio

R is a programming language and totally free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and other analysis for date sets. R is highly extensible through the use of user-submitted packages for specific functions or special areas of study. Due to its S heritage, R has stronger object-oriented programming facilities than the other statistical computing languages. Extending R is also eased by its lexical scoping rules. Another strength of R is static graphics, which can produce publication-quality graphs, including mathematical symbols. Dynamic and interactive graphics are available through additional packages.

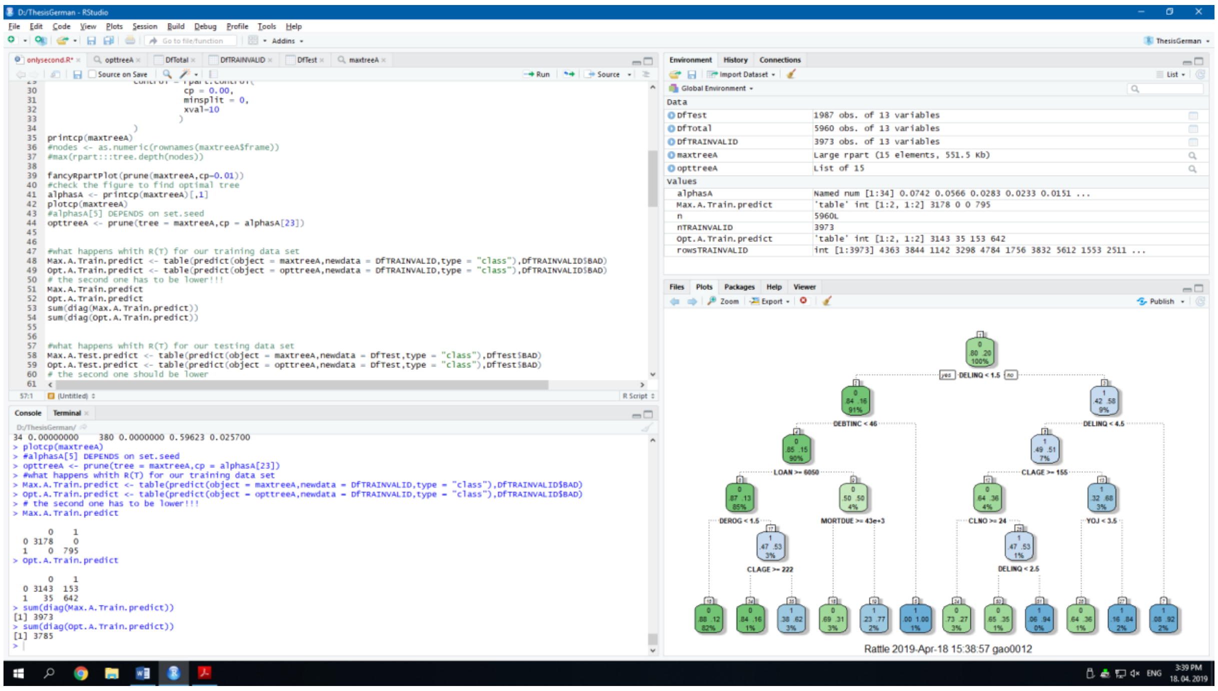
. R Studio is a free and open-source integrated development environment for R, a programming language for statistical computing and graphics. R Studio was founded by JJ Allaire, creator of the programming language ColdFusion. Hadley Wickham is the Chief Scientist at R Studio. That is, to build a classification tree and calculate prediction results.

We can download R and R Studio from internet.

(https://cran.r-project.org/ and https://www.rstudio.com/products/rstudio/download/)

A screenshot from the R studio is shown in Fig 3.1.

Figure 3.1 The screenshot of the R studio



```
                        cp = 0.00,
                        minsplit = 0,
                        xval=10
                       )
              )
printcp(maxtreeA)
#nodes <- as.numeric(rownames(maxtreeA$frame))
#max(rpart:::tree.depth(nodes))

fancyRpartPlot(prune(maxtreeA,cp=0.01))
#check the figure to find optimal tree
alphasA <- printcp(maxtreeA)[,1]
plotcp(maxtreeA)
#alphasA[5] DEPENDS on set.seed
opttreeA <- prune(tree = maxtreeA,cp = alphasA[23])


#what happens whith R(T) for our training data set
Max.A.Train.predict <- table(predict(object = maxtreeA,newdata = DfTRAINVALID,type = "class"),DfTRAINVALID$BAD)
Opt.A.Train.predict <- table(predict(object = opttreeA,newdata = DfTRAINVALID,type = "class"),DfTRAINVALID$BAD)
# the second one has to be lower!!!
Max.A.Train.predict
Opt.A.Train.predict
sum(diag(Max.A.Train.predict))
sum(diag(Opt.A.Train.predict))


#what happens whith R(T) for our testing data set
Max.A.Test.predict <- table(predict(object = maxtreeA,newdata = DfTest,type = "class"),DfTest$BAD)
Opt.A.Test.predict <- table(predict(object = opttreeA,newdata = DfTest,type = "class"),DfTest$BAD)
# the second one should be lower
```

```
34 0.00000000    380 0.0000000 0.59623 0.025700
> plotcp(maxtreeA)
> #alphasA[5] DEPENDS on set.seed
> opttreeA <- prune(tree = maxtreeA,cp = alphasA[23])
> #what happens whith R(T) for our training data set
> Max.A.Train.predict <- table(predict(object = maxtreeA,newdata = DfTRAINVALID,type = "class"),DfTRAINVALID$BAD)
> Opt.A.Train.predict <- table(predict(object = opttreeA,newdata = DfTRAINVALID,type = "class"),DfTRAINVALID$BAD)
> # the second one has to be lower!!!
> Max.A.Train.predict

       0    1
  0 3178    0
  1    0  795
> Opt.A.Train.predict

       0    1
  0 3143  153
  1   35  642
> sum(diag(Max.A.Train.predict))
[1] 3973
> sum(diag(Opt.A.Train.predict))
[1] 3785
>
```

```
Data
DfTest                1987 obs. of 13 variables
DfTotal               5960 obs. of 13 variables
DfTRAINVALID          3973 obs. of 13 variables
maxtreeA              Large rpart (15 elements, 551.5 Kb)
opttreeA              List of 15
Values
  alphasA             Named num [1:34] 0.0742 0.0566 0.0283 0.0233 0.0151 ...
  Max.A.Train.predict 'table' int [1:2, 1:2] 3178 0 0 795
  n                   5960L
  nTRAINVALID         3973
  Opt.A.Train.predict 'table' int [1:2, 1:2] 3143 35 153 642
  rowsTRAINVALID      int [1:3973] 4363 3844 1142 3298 4784 1756 3832 5612 1553 2511 ...
```

The upper left window is a plane text editor: the source editor. This is where we write the code. Our code will not be evaluated until we "run" them to console. The area under the source editor is console, where our code from the source is evaluated by R. we can also use the console to preform quick calculations that we don't need to save. The upper right is the environment/ history. Here we can see what objects are in our working space (environment) or view our command history. The last one is file/ plots/ packages / help. Here we can see file directories, view plots, download packages and access R help. In general, if we want to use any non-standard packages in R studio, like rpart we have to install the package

```
install.packages("rpart")
```

and then attach the package

```
library("rpart")
```

Note that function library is possibly the most common function call in R.

3.2 Package **rpart**

In 1999, R package rpart (algorithm of Recursive trees) was the first time introduced to the public (Therneau and Atkinson, 1997). You can find more information including reference manual (https://cran.r-project.org/web/packages/rpart/index.html). And it is continuously updated. The section is intended to give a short overview of the methods found in the rpart routines, which implement many of the ideas found in the CART book and programs of Breiman, Friedman, Olshen and Stone (1993).

The data we use in the application part is csv format. It can be imported to R studio as a data frame object. We can sue the function read.csv for this purpose. We use package rpart to analyze the data set in terms of this command, namely to find a maximal tree:

```
    maxtree <- rpart( formula= Y ~ .,
                      data=MYDATA,
                      method = "class",
                      parms = list(prior = c(N1/N,N2/N),
                                   split = "gini",
                                   loss=matrix(c(0,1,1,0),
ncol=2)
                                   ),
                      control = rpart.control(minsplit =0,
                                              cp=0.00,
                                              xval=10)
```

)

Let me describe the parameters involved in `rpart` now, I will use the notation introduced in Chapter 2 to explain the `rpart`. There a lot of functions in this package. We will use the following function: `rpart, rpart.control, prune.rpart, summary.raprt, predict.rpart, printcp, plot.rpart, plotcp.`

Parameters appearing in `rpart` function:

•`data`: `data` parameter is a data frame object with our training data set.

•`formula`: It represents our model, as we want to classify Y according to $X_1,\ldots, X_n$ we use formula = $Y\sim X_1+\ldots+X_n$, or simply formula=Y~. If `data` contains only predictor Y and independent variables $X_1,\ldots, X_n$.

•`method`: The splitting rule depends on a type of a tree model. We can build trees based upon "anova" (regression), "poisson" or "exp" approach. If method is missing then the routine tries to make an intelligent guess. If Y is a factor then `method = "class"` is assumed, otherwise `method = "anova"` is assumed.

•`parms`

➢ `prior`: For classification splitting, the list can contain any of: the vector of prior probabilities π_i(component prior) .The priors must be positive and sum to 1and the default priors are proportional to the data counts N_1/N N_2/N.

➢ `split`: The splitting index can be Gini or information. The losses default to 1, and the split defaults to Gini. In the thesis, I only use the Gini index (2.3.1) to describe the data.

➤ `loss`: Loss=matrix(c(0,1,1,0), ncol=2). The command matrix (c(0,1,1,0), ncol=2) represents the 2×2 matrix $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. The loss matrix introduced in Chapter 2.

When we want to get different misclassification of class 1 as class 2 and class 2 as class 1, in terms of function $C(i|j)$ (2.1). In general, a loss matrix L (component loss) which represents penalty has a form of $\begin{pmatrix} C(1|1) & C(1|2) \\ C(2|1) & C(2|2) \end{pmatrix}$. It must have zeros on the diagonal and positive off-diagonal elements.

•control

➤ `minsplit`: The minimum number of observations in a node for which the routine will even try to compute a split. And the minimum number of observations that must exist in a node in order for a split to be attempted. That is, if we want to build a maximal tree we should set minisplit= 0.

➤ `cp`: `cp` represents α in formula (2.2). It already control the pruning phase. As mentioned above, so we have to set `cp` =0 to obtain the maximal tree.

➤ `xval`: The number of cross-validations. In the thesis, we use 10-fold cross validation, so we set xval=10.

•`printcp`: Displays the cp table for fitted itree object. Using this command, we would get table which summarize the pruning phase graphs with data generated by V-fold cross-validation function.

•`plotcp`: It provides a graphical representation to the cross validated error summary which is output of `printcp` function. The `cp` values are plotted against the geometric mean to depict the deviation until the minimum value is reached.

•`prune`: It allows us to generate a subtree for given value of `cp`.

•`summary`: It summarizes the output or `rpart` function.

•`predict`: It is used to predict values of given target variable data set. In the command, we only write the object and the data set based on.

3.3 Other used packages and functions

•`rattle`: To draw trees we use rattle package, for root node we use `rpart.plot`. `Rattle` is a package written in R providing a graphical user interface to very many other R packages that provide functionality for data mining.

•`sample`: Takes a sample of the specified size from the elements of X using either with or without replacement.

•`set.seed`: Set the seed of R's random number generator, which is useful for creating simulations or random objects that can be reproduced. In the application, we use `set.seed(160)` to select the specific data set and make the whole process reproducible.

4. Application for Classification and Prediction

Credit scoring, also known as credit rating and credit evaluation, is a social intermediary service that provides credit information to the society or provides decision-making reference for the unit itself. Originally produced in the United States in the early 20th century. Later extended to a variety of financial products and various assessment objects. Due to the different objects and requirements of credit scorings, the content and methods of credit scoring are also quite different. The credit scoring is an individual or company financial history that indicates whether the person or company can repay the debt, based on the amount and whether the debtor repaid the previous debt in a timely manner.

In this chapter, we will use the method we described to analysis the data sets and show the result of prediction.

4.1 Credit scoring

Credit-scoring techniques assess the risk in lending to a special consumer. It is an evaluation by a lender of a borrower and incarnates the circumstances of both and the lender’s view of the possibly future financial scenarios. Some lenders will evaluate an individual as creditworthy and others will not. One of the perennial dangers of credit scoring is that this may cause to be the proposal, and there will be those who can get credit from all financial intermediaries and those who cannot. Explaining someone as uncreditworthy causes offense. It will be better for the lender to describe the reality, which is that the position of lending to this consumer represents a risk that the lender is not willing to deal.

Credit scoring have two definitions, narrow and broad. The narrow credit scoring refers to an independent third-party credit scoring agency that evaluates the creditor's ability and willingness to repay the principal and interest of the debt in full, and uses a simple scoring symbol to indicate its default risk and the severity of the loss. The broad credit scoring is an overall assessment of the ability and willingness of the scoring object to perform relevant contracts and economic commitments.

Credit scoring is an inevitable outcome of the development of the market economy. From the perspective of economics, the theory of information asymmetry lays a theoretical foundation for the emergence of credit evaluation. The problem of information asymmetry exists in a large number of transactions between banks and enterprises, between enterprises, and in the capital market. In order to reduce the harm caused by information asymmetry - credit risk, credit scoring came into being. Due to the continuous development of the market economy, enterprises began to invest in the capital market in order to achieve development. There are a wide range of information asymmetry in the process of bank credit, commercial transaction and capital market transactions. Through credit evaluation, qualitative analysis, quantitative calculation and measurement of the default probability of borrowers may be made. Asymmetry can play an important role.

The credit scoring is essentially a way of dividing the borrower into different groups according to its characteristics and assessing the credit of different groups. In the late 1960s, the emergence of credit cards made banks and other credit card issuers aware of the importance of credit scoring. The surge in the number of people applying for credit cards every day makes it impossible to manually analyze the credit risk of applicants in terms of both economic and human resources, so the automated loan review technology came into being. Using credit scores, financial institutions have found that it can better predict default rates and reduce loan default rates by 50% or more.

In the 1980s, the success of credit scorecards prompted banks to apply the technology extensively to personal consumer loans, home loans, and small business loans. In credit card modeling techniques, Logistic regression models and linear programming models are undoubtedly the two pillars of traditional methods. In recent years, with the advancement of technology, analytical methods such as artificial intelligence technologies, such as expert systems and neural networks, have emerged and are being introduced into credit card modeling. The focus of international credit research has shifted from looking for the smallest data base with default rates to looking for profit-maximizing customer groups.

The credit scoring was originally judged manually by the loan officer. Credit analysts read the applicant form and classify the customer, and the classification basis is usually 3C, 4C or 5C: characteristics of individuals and family members, asset status, collateral, income ability, market environment, etc. The current credit score is based on statistical methods, using linear regression, and with the help of computers, mathematically classifying customers, such as logistic regression and classification tree models. Most scorecard modelers use one or a combination of technologies.

In all cases, whatever the techniques used, the important point is that there is a very huge sample of previous customers with their application details and consecutive credit history available. All the techniques use the sample to recognize the connections between the proportion of the consumers and how “good” or “bad” their consecutive history is. Many of the methods lead to a scorecard, where the characteristics are given a score and the total of these points says whether the risk of a consumer being bad is too great to accept.

It can be said that credit means risk, and the risk of credit is generated along with the generation of credit activities. The analysis and evaluation of credit risk have also undergone profound changes: from empirical judgment to index calculation, from qualitative argumentation to quantitative analysis. Generally calculated to the digital model judgment and so on. Credit risk assessment can be divided into two main categories: factor analysis and model analysis.

4.2 Description of data

The Home Equity dataset (HMEQ) contains baseline and loan performance information for 5,960 recent home equity loans. A home equity loan is a loan where the obligor uses the equity of his or her home as the underlying collateral. The target (BAD) is a binary variable indicating whether an applicant eventually defaulted or was seriously delinquent. This adverse outcome occurred in 1,189 cases (19.94%).

(Source: https://www.kaggle.com/ajay1735/hmeq-data)

For each applicant, 12 input variables were recorded, we can see the detail in the Table 4.1.

- BAD: 0 = applicant paid loan (Good applicant); 1 = applicant defaulted on loan or seriously delinquent (Bad applicant)
- LOAN: Amount of the loan request, it ranges from 1100 to 89900 (numerical variable)
- MORTDUE: Amount due on existing mortgage ,5053 unique values (numerical variable)
- VALUE: Value of current property, 5381 unique values (numerical variable)
- REASON: DebtCon = debt consolidation; HomeImp = home improvement
- JOB: Occupational categories (other 40%, profExe 21%, office 16%, Mgr 13%, the other 10%)
- YOJ: Years at present job, it ranges from 0 to 26 years
- DEROG: Number of major derogatory reports (0 76%, 1 7%, the other 17%)
- DELINQ: Number of delinquent credit lines (0 70%, 1 11%, the other 19%)

- CLAGE: Age of oldest credit line in months, 5314 unique values (numerical variance)
- NINQ: Number of recent credit inquiries
- CLNO: Number of credit lines
- DEBTINC: 4693 unique values (numerical variable)

We can use function `read.csv` to read the whole data as data frame `Dftotal`.

```
DfTotal <- read.csv("./Data/hmeq.csv")
```

Table 4.1 HEMQ data sets (first 30 observations)

number	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
1	1	1100	25860	39025	HomeImp	Other	10.5	0	0	94.36667	1	9	
2	1	1300	70053	68400	HomeImp	Other	7	0	2	121.8333	0	14	
3	1	1500	13500	16700	HomeImp	Other	4	0	0	149.4667	1	10	
4	1	1500											
5	0	1700	97800	112000	HomeImp	Office	3	0	0	93.33333	0	14	
6	1	1700	30548	40320	HomeImp	Other	9	0	0	101.466	1	8	37.11361
7	1	1800	48649	57037	HomeImp	Other	5	3	2	77.1	1	17	
8	1	1800	28502	43034	HomeImp	Other	11	0	0	88.76603	0	8	36.88489
9	1	2000	32700	46740	HomeImp	Other	3	0	2	216.9333	1	12	
10	1	2000		62250	HomeImp	Sales	16	0	0	115.8	0	13	
11	1	2000	22608				18						
12	1	2000	20627	29800	HomeImp	Office	11	0	1	122.5333	1	9	
13	1	2000	45000	55000	HomeImp	Other	3	0	0	86.06667	2	25	
14	0	2000	64536	87400		Mgr	2.5	0	0	147.1333	0	24	
15	1	2100	71000	83850	HomeImp	Other	8	0	1	123	0	16	
16	1	2200	24280	34687	HomeImp	Other		0	1	300.8667	0	8	
17	1	2200	90957	102600	HomeImp	Mgr	7	2	6	122.9	1	22	
18	1	2200	23030				19						3.711312
19	1	2300	28192	40150	HomeImp	Other	4.5	0	0	54.6	1	16	
20	0	2300	102370	120953	HomeImp	Office	2	0	0	90.99253	0	13	31.5885
21	1	2300	37626	46200	HomeImp	Other	3	0	1	122.2667	1	14	
22	1	2400	50000	73395	HomeImp	ProfExe	5	1	0		1	0	
23	1	2400	28000	40800	HomeImp	Mgr	12	0	0	67.2	2	22	
24	1	2400	18000		HomeImp	Mgr	22		2	121.7333	0	10	
25	1	2400		17180	HomeImp	Other		0	0	14.56667	3	4	
26	1	2400	34863	47471	HomeImp	Mgr	12	0	0	70.49108	1	21	38.2636
27	0	2400	98449	117195	HomeImp	Office	4	0	0	93.81177	0	13	29.68183
28	1	2500	15000	20200	HomeImp		18	0	0	136.0667	1	19	
29	1	2500	25116	36350	HomeImp	Other	10	1	2	276.9667	0	9	
30	0	2500	7229	44516	HomeImp	Self		0	0	208	0	12	

As you can see in Tab 4.1, our data set classify missing values. But because CART algorithm is an algorithm that handle missing values we do not need to delete lanes with missing values. There can be also work in the model building (see the details in the Breiman). We need to divide the data set in two parts randomly. There are 5,970 applicants in the entire data set. According to the principle of separate data, it is divided into training data set (2/3 of total) and testing data set (1/3 of total). In the training data set, 3178 applicants (79.98%) were in class 0 (good applicant) and 795 applicants (20.02%) were in class 1 (bad applicant). In the testing data set, 1592 applicants (80.12%) were in class 0 (good applicant) and 395 applicants (19.88%) were in class 1 (bad applicant).

For choosing the data group we need to set the random generator

```
set.seed(160)
```

and so choose rows representing our training data set:

```
rowsTRAINVALID <- sample(x = 1:5970,size = 3973)
```

And then we assign our training data set to variable `DfTRAINVALID` as follows

```
DfTRAINVALID <- DfTotal[rowsTRAINVALID,]
```

Then we assign the testing data set to variable `DfTest`

```
DfTest <- DfTotal[-rowsTRAINVALID,]
```

Table 4.2 Frequencies and percentages

	0 (Good)	1 (Bad)	Total
Training	3178 (79.98%)	795 (20.02%)	3973
Testing	1592 (80.12%)	395 (19.88%)	1987
Total	4770 (79.89%)	1200 (20.11%)	5970

We want to find out how the target variable BAD can be predicted by values of 11 independent variables (LOAN, MORTDUE, VALUE, REASON, JOB, YOJ, DEROG, DELINQ, CLAGE, NINQ, CLNO, DEBTINC).

4.3 Growing phase

In this chapter, we will use R studio to generate classification trees.

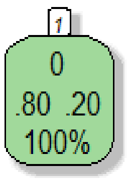
4.3.1 Root tree

Before we introduce the maximal tree generated by rpart package let us describe how the first splits are generated to better understand the idea of recursive partitioning method. As I mentioned in Chapter 3. We need to change the parameter in `rpart` function. That is, we set the cp=0 and minsplit=0 to build the maximal tree.

```
maxtreeA <- rpart(formula = BAD~.,
                  data = DfTRAINVALID,
                  method = "class",
                  parms = list(
                         split = "gini",
                         loss=matrix(c(0,1,1,0),ncol=2)
                         ),
                  control = rpart.control(
                           cp = 0.0,
                           minsplit = 0,
                           xval=10
                          )
                  )
```

Note that the new setting may not ensure that the maximal tree in found, since the depth of `rpart` package generated tree in limited to 30. We describe the original data set is nothing but a root tree T_{root}, consisting of one node without any splits. Classification and regression trees can be generated in R with `rpart` package. We can draw (small) decision trees with packages `rpart.plot` and `rattle`. Here is the root tree in terms of proportions and frequencies.

Figure 4.1 Root tree



Let me denote the node by A_1 and use the follow notation (similar notation will be applied for all the nodes):

n_{A_1}, the number of observations in node A_1.

$p(A_1) = 1 = 100\%$, probability of A_1 (for future observations); this node includes 100% of the data.

$p(1|A_1) = 0.2002 = 795/3973$, probability of applicant defaulted on loan or seriously delinquent in node A_1 (for future observation).

$p(0|A_1) = 0.7998 = 3178/3973$, probability of applicant paid loan in node A_1 (for future observations).

Because there is no rule, we can see the whole node classified as class 0. $d(A_1) = 0$, the class assigned to A_1, which is nothing but the modal class. And 79.98% is the correct, so the misclassification rate $R(T_{root}) = 20.02\%$.

If a node is a pure node, then its risk is zero. When I look for a rule to split the node I want its sons to be as pure as possible. I apply Gini impurity function in the following form to quantify purity of a node:

$$I(A) = p(1|A) \cdot (1 - p(1|A)) + p(0|A) \cdot (1 - p(0|A)) =$$

$$= 1 - p(1|A)^2 - p(0|A)^2.$$

That is,

$$I(A_1) = 1 - p(1|A_1)^2 - p(0|A_1)^2 = 1 - (0.7998)^2 - (0.2002)^2 = 0.3202.$$

4.3.2 First split

If I use the second row in the Tab 4.11, when I set the cp=0.06—we will describe it in detail in Chapter 3—then I can get the first split tree. With the help of function `prune()`,

```
firsttree <- prune(tree = maxtreeA,cp = 0.06)
```

The first tree is relatively simple, so we can use command `summary(firsttree)`

to see detail of the first tree (Fig 4.3)

We can draw the tree with function `fancyrpartplot()`, which is nothing but graphs of R object `firsttree`:

```
n= 3973

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 3973 795 0 (0.7998993 0.2001007)
  2) DELINQ< 1.5 3610 584 0 (0.8382271 0.1617729) *
  3) DELINQ>=1.5 363 152 1 (0.4187328 0.5812672) *
```

We have to choose a rule for the first split now. We can use 11 different rules for the first binary split. Let me start with variables with DELINQ which depends on whether it is less than 1.5 which means the number are 0 and 1. In the Figure 4.2, 91% of the total are less than 1.5 which represents the integers more than 1. The other (9%) are more than 1.5. In the node 2, I use the DEBTINC as the judgement.

Then in the node 2 and node 3, we can get the minimal amount of

$$0.91(1 - 0.84^2 - 0.16^2) + 0.09(1 - 0.42^2 - 0.58^2) = 0.2885$$

That is, in this first tree, I can achieve lower impurity because of $0.32 > 0.2885$. But if I set other different rule for the split. Then I can't get the results the impurity lower than 0.32. So the $DELINQ < 1.5$ is the rule to split.

Figure 4.2 The first tree

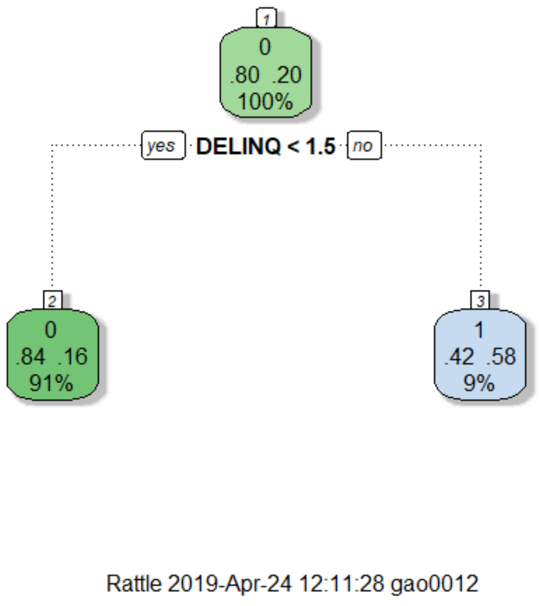


Figure 4.3 Description of first tree

```
rpart(formula = BAD ~ ., data = DfTRAINVALID, method = "class",
    parms = list(split = "gini", loss = matrix(c(0, 1, 1, 0),
        ncol = 2)), control = rpart.control(cp = 0, minsplit = 0,
        xval = 10))
  n= 3973

          CP nsplit rel error    xerror       xstd
1 0.07421384      0 1.0000000 1.0000000 0.03172007
2 0.05660377      1 0.9257862 0.9710692 0.03137076

Variable importance
DELINQ    CLNO
    99       1

Node number 1: 3973 observations,    complexity param=0.07421384
  predicted class=0  expected loss=0.2001007  P(node) =1
    class counts:  3178   795
   probabilities: 0.800 0.200
  left son=2 (3610 obs) right son=3 (363 obs)
  Primary splits:
      DELINQ  < 1.5      to the left,  improve=111.10760, (400 missing)
      DEBTINC < 43.99281 to the left,  improve= 94.72427, (871 missing)
      DEROG   < 0.5      to the left,  improve= 74.70250, (495 missing)
      LOAN    < 6050     to the right, improve= 55.18328, (0 missing)
      CLAGE   < 172.5656 to the right, improve= 43.35843, (211 missing)
  Surrogate splits:
      CLNO < 64.5     to the left,  agree=0.899, adj=0.008, (247 split)

Node number 2: 3610 observations
  predicted class=0  expected loss=0.1617729  P(node) =0.9086333
    class counts:  3026   584
   probabilities: 0.838 0.162

Node number 3: 363 observations
  predicted class=1  expected loss=0.4187328  P(node) =0.09136673
    class counts:   152   211
   probabilities: 0.419 0.581
```

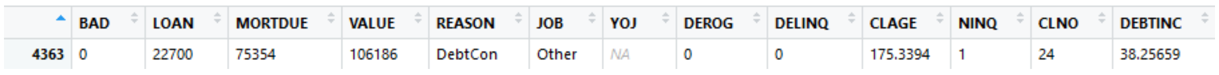
- **Prediction of single case**

We can already use this small tree for classification. As mentioned in Chapter 3, we use function `predict()` for this purpose. We show and describe the use of a tree-based classification for one particular case from the data set.

```
Dfonecase <- DfTRAINVALID[1,]
```

Using the above formula to get one row of data set,

Table 4.3 First row of `DfTrainvalid` data set



	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
4363	0	22700	75354	106186	DebtCon	Other	NA	0	0	175.3394	1	24	38.25659

The first number is the order of the data. Next is the 12 variables of this applicant. Then we need to predict if the applicant can pay the loan. Firstly, we set parameter `type` to “`class`”.

```
predict(object = firsttree,newdata = Dfonecase ,type =
"class")
```

Secondly, if we set parameter `type= "prior"`, we can see also the probability of both classes for this single case.

```
predict(object = firsttree,newdata = Dfonecase ,type =
"prior")
```

Table 4.4 Prediction of first row

0 (Good)	1 (Bad)
0.8382	0.1618

As we can see in the Tab 4.4, the percentage of 83.82% classified as class 0, which represents a high probability that the applicant is a good client. The percentage of 16.18% classified as class 1, which represents a low probability that the applicant is a bad client.

That is how we can apply this method to our data set. The next few commands are to classify and predict the first tree.

```
predict(object = firsttree,newdata = DfTRAINVALID)

table(predict(object = firsttree,newdata =
DfTRAINVALID,))
```

And summarize it in Tab 4.5.

We can get such output. There are 3610 applicants classified as class 0 (good applicant), they account for 90.86% of the total. There are 363 applicants classified as class 1(bad applicant), they account for 9.14% of the total.

Table 4.5 The output of first prediction

	0 (Good)	1 (Bad)
Number	3610	363
Proportion	0.9086	0.0914
Percent	90.86%	9.14%

The result of the prediction is actually the data in the Node 2 and Node 3 (Fig 4.2).

After this, we can calculate misclassification rate after first split for the training data set.

```
Max.A.Train.predict <- table(predict(object = 
firsttree,newdata = DfTRAINVALID,type = 
"class"),DfTRAINVALID$BAD)
```

We use our training data `trainvaild` with 3973 observations. There are 3178 applicants who can pay the loan and 795 applicants who cannot pay the loan in truth. When we use the command to predict whether they will be able to pay the loan or not we classify 3610 applicant of them as being able to pay the loan and 363 applicants as not being able to pay the loan. The summary of this prediction can be seen in Tab 4.6.

Table 4.6 Misclassification matrix

		Actual Values		
		0 (Good)	1 (Bad)	Total
Predicted	0 (Good)	3026 (76.16%)	584 (14.70%)	3610 (90.86%)
	1 (Bad)	152 (3.83%)	211 (5.31%)	363 (9.14%)
	Total	3178 (79.99%)	795 (20.01%)	3973 (100%)

The table includes cross-tabulation of predicted values of variable BAD and actual values of this variable. We can see that 3026+211 = 3237 applicants were classified correctly. On the other hand, we have 584 applicants who cannot pay the loan but we classify them as being able and 152 applicants who are able to pay the loan and we classify them as bad applicants. The misclassification error of this classification is

$$R(T_{first}) = \frac{584 + 152}{3026 + 584 + 152 + 211} = 18.53\%$$

and so the accuracy of this classification is 100% - 18.53%=81.47%.

4.3.3 Second split

That is, we use the same idea to apply for the second split. By using the similar formulas,

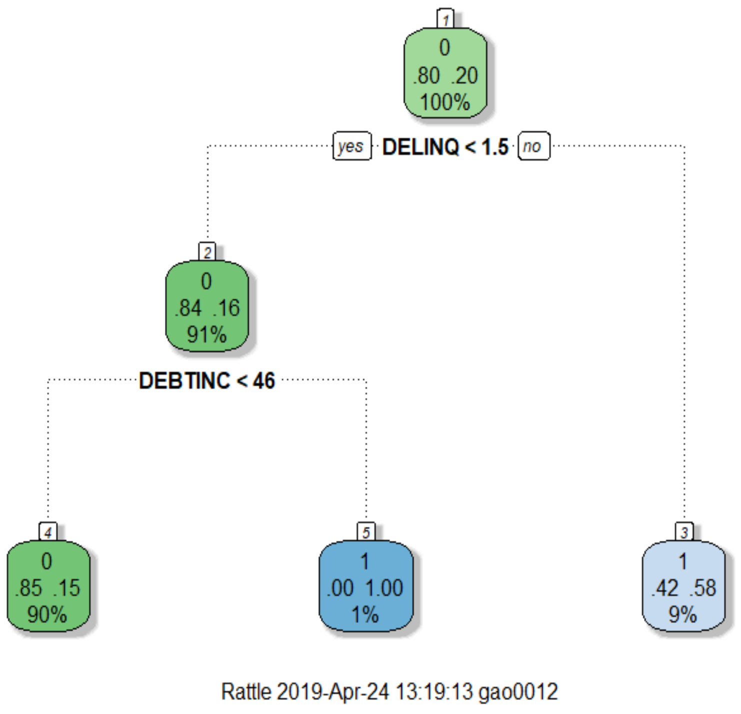
```
secondtree <- prune(tree = maxtreeA,cp = 0.04)

fancyRpartPlot(secondtree)

predict(object = secondtree,newdata = DfTRAINVALID)
```

```
    table(predict(object    =    secondtree,newdata    =
DfTRAINVALID,type = "class"))
```

Figure 4.4 The second split



It can be seen that in the second tree we have added a judgment path (DEBTINC < 46). When the judgement path is increased, there is no doubt that this will make the analysis more accurate.

Table 4.7 Prediction after second split

	0 (Good)	1 (Bad)
Number	3565	408
Proportion	0.8973	0.1027
Percent	89.73%	10.27%

There are 3565 applicants classified as class 0 (good applicant), they account for 89.73% of the total. There are 408 applicants classified as class 1(bad applicant), they account for 10.27% of the total.

For this condition, we need to know what the misclassification rate is, we still use our training data `trainvaild` with 3973 observations. There are 3178 applicants who can pay the loan and 795 applicants who cannot pay the loan in truth. We classify 3565 applicant of them as being able to pay the loan and 408 applicants as not being able to pay the loan. The summary of this prediction can be seen in Tab 4.8.

Table 4.8 Misclassification matrix

		Actual Values		
		0 (Good)	1 (Bad)	Total
Predicted	0 (Good)	3026 (76.16%)	539 (13.57%)	3565 (89.73%)
	1 (Bad)	152 (3.83%)	256 (6.44%)	408 (9.27%)
	Total	3178 (79.99%)	795 (20.01%)	3973 (100%)

The table includes cross-tabulation of predicted values of variable BAD and actual values of this variable. We can see that 3026+256 = 3282 applicants were classified correctly. On the other hand, we have 539 applicants who cannot pay the loan but we classify them as being able and 152 applicants who are able to pay the loan and we classify them as bad applicants. The misclassification error of this classification is

$$R(T_{second}) = \frac{539 + 152}{3026 + 539 + 152 + 256} = 17.39\%$$

So the accuracy of this classification is 100% - 17.39%=82.61%. It can be seen that the accuracy of the second tree is increased 82.61%-81.47%= 1.14%compared to the first tree. Later we will use function `predict()` to calculate misclassification errors for even more complex trees.

4.3.4 Maximal tree

There are 761 leaf nodes in the maximal tree. `nsplit` represents the number of splits. It is in the Table 4.10 when real error is 0. The `nsplit` for the maximal tree is 380. And there are 796 nodes of the maximal tree. The height of this tree is 28. It represents the maximal number of questions for the clients to classify her or him.

Now we show the maximal tree performance evaluation shown in. We use our training data `Trainvaild` with 3973 observations for this purpose. There are 3178 applicants who can pay the loan and 795 applicants who cannot pay the loan in truth. We create and describe the confusion matrix for prediction with maximal tree now. When we use the model found in previous subchapter to predict whether they will be able to pay the loan or not we classify 3178 applicant of them as being able to pay the loan and 795 applicants as not being able to pay the loan. The summary of this prediction can be seen in Tab 4.9.

Table 4.9 Maximal tree prediction of train data set

		Actual Values		
		0 (Good)	1 (Bad)	Total
Predicted	0 (Good)	3178 (79.99%)	0 (0%)	3178 (79.99%)
	1 (Bad)	0 (0%)	795 (20.01%)	795 (20.01%)
	Total	3178 (79.99%)	795 (20.01%)	3973 (100%)

The table includes cross-tabulation of predicted values of variable BAD and actual values of this variable. We can see that 3178+795 = 3973 applicants were classified correctly. This is just the whole of training data set. That's why the misclassification error of this classification is

$$R(T_{max}) = \frac{0+0}{3178+0+0+795} = 0$$

and the accuracy of this classification is 100%. Obviously, it can be concluded that the maximum tree is much more accurate than the previous example of the first two trees. Because the largest tree has added all the judgment conditions, so there is no misclassification error.

4.4 Pruning phase

The next step in the process is pruning the tree in an attempt to avoid overfitting. The pruning process works upwards through the partition nodes from the bottom of the tree. The decision of whether to prune a node is controlled by the complexity parameter, introduced in α, which balances the additional complexity of adding the split at the node against the increase in predictive accuracy that it offers. A higher complexity parameter leads to more and more nodes being pruned off, resulting in smaller trees.

Table 4.10 The cp table generated by `printcp`

	cp	nsplit	rel error	xerror	xstd
1	0.07421384	0	1.000000	1.00000	0.031720
2	0.05660377	1	0.925786	0.97107	0.031371
3	0.02830189	2	0.869182	0.90189	0.030491
.					
.					
20	0.00287511	82	0.357233	0.61887	0.026116
21	0.00251572	92	0.325786	0.56855	0.025175
22	0.00188679	110	0.280503	0.55472	0.024906
23	**0.00167715**	**132**	**0.236478**	**0.54717**	**0.024757**
24	0.00125786	135	0.231447	0.55346	0.024881
.					
.					
32	0.00050314	363	0.008805	0.59119	0.025606
33	0.00041929	372	0.003774	0.59623	0.025700
34	0.00000000	380	0.000000	0.59623	0.025700

Table 4.11 Adjusted cp table (in terms of α)

	α	$\|T\|$	$R(\alpha)$	$R^{CV}(\beta)$	$SE^{CV}(\beta)$
1	0.01485024	1	0.200101	0.20010	0.006347
2	0.01132645	2	0.185250	0.19431	0.006277
3	0.00566323	3	0.173924	0.18047	0.006101
.					
.					
20	0.00057531	83	0.071483	0.12384	0.005226
21	0.00050340	93	0.065190	0.11377	0.005038
22	0.00037755	111	0.056129	0.11100	0.004984
23	**0.00033560**	**133**	**0.047319**	**0.10949**	**0.004954**
24	0.00025170	136	0.046313	0.11075	0.004979
.					
.					
32	0.00010068	364	0.001762	0.11830	0.005124
33	0.00008390	373	0.000755	0.11931	0.005143
34	0.00000000	381	0.000000	0.11931	0.005143

You can see the whole tab 4.10 in annexes.

In the Tab 4.10. The first column is the order of a tree. The second column is the cp parameter used in function `rpart` introduced in Chapter 3. When we multiply it by root node error which is 795/3973, we can get the complexity parameter α (function

2.2). The third column is number of the splits which means the splits of the trees. The fourth, fifth and sixth column multiplied by root node error are a misclassification rate from the learning data set $R(\alpha)$, mean misclassification rate generated by cross validation $\mathrm{R}^{CV}(\beta)$ and its standard error $\mathrm{SE}^{CV}(\beta)$.

The first row represents our root node. As we mentioned above when we set cp to any value from the interval [0.0742, ∞) then we generate the tree without any splits. In terms of complexity parameter $\alpha \in [0.0149, \infty)$.

In the second row, when we set cp ∈ [0.0566, 0.0742), in terms of complexity parameter $\alpha \in [0.0113, 0.0148)$. I can get the tree with only one split.

In the third row, when we set cp ∈ [0.0283, 0.0566), in terms of complexity parameter $\alpha \in [0.0057, 0.0113)$. I can get the second split.

Similarly, in the 34th row, when we set cp ∈ [0, 0.0004), in terms of complexity parameter $\alpha \in [0, 0.00008)$. We can simply set α=0 to get the maximal tree with 381 terminal nodes.

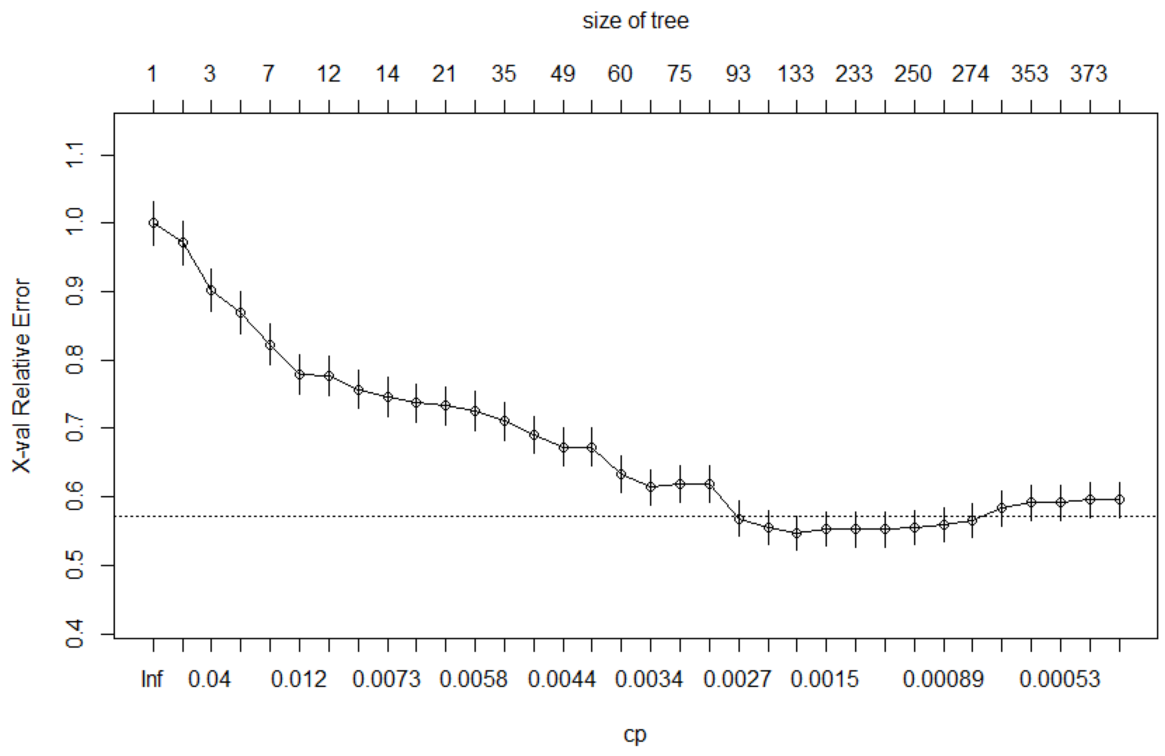
4.5 10 fold cross-validation

As I described in Table 2.1, I chosen 10 subsets. 9 of them are training data sets, the other one is testing data set. We are looking for β_{min} with the smallest cross-validation error $\mathrm{R}^{\mathrm{CV}}(\beta_{min})$. If we combine the Table 4.10 and Figure 4.5, we can choose the 23th as the minimal error tree. That is, when we take $\alpha \in [0.0017, 0.0019)$. We can generate the tree with 133 terminal nodes. As I described in the chapter 2.4.2, we have to choose a representative from the interval, namely, square root of a product of the limits:

$$\beta_{23} = \sqrt{\alpha_{23}\alpha_{22}} = \sqrt{0.00167715 \cdot 0.00188679} = 0.0018.$$

The lower horizontal axis represents the value of cp which is xerror in Tab 4.10. The lower horizontal is the size of trees which is the $|T|$ in Tab 4.11. And the vertical axis represents X-val Relative Error. The 23rd point is smaller than the value represented by the dotted line for the first time, that is, we focus on the 23rd tree.

Figure 4.5 plotcp output



Then use the function,

$$R^{CV}(\beta_{23}) = \frac{1}{10}\sum_{i=1}^{10} T^{(i)}(\beta) = 0.54717 * \left(\frac{795}{3973}\right) = \frac{435}{3973} = 0.1095.$$

Which is $R^{CV}(\beta)$ shown in Tab 4.11.

$$SE^{CV}(\beta_{23}) = \sqrt{\frac{R^{CV}(\beta)\left(1 - R^{CV}(\beta)\right)}{N}} = \sqrt{\frac{(\frac{435}{3973})(1-\frac{435}{3973})}{3973}} = 0.0050.$$

Which is $SE^{CV}(\beta)$ shown in Tab 4.11.

Note that choice of β_{min} can be affected by the seed of the random number generator to separate learning data set into 10 subtest. Small changes can be large changes in number of terminal nodes in optimal tree. To choose the simplest tree whose accuracy is comparable to $R^{cv}(\beta_{\text{min}})$, 1-SE rule is applied to select the right sized tree: An optimal tree, T (β_{opt}), is the tree corresponding to β_{opt} where β_{opt} is the maximum β satisfying

$$\mathrm{R}^{CV}(\beta) \leq \mathrm{R}^{CV}(\beta_{min}) + \mathrm{SE}^{CV}(\beta_{min})$$

In this output,

$$\mathrm{R}^{CV}(\beta_{min}) + \mathrm{SE}^{CV}(\beta_{min}) = 0.1144$$

If we look at the 20th, 21st, 22nd, 23rd and 24th row,

$$\mathrm{R}^{CV}(\beta_{20}) = 0.12384$$

$$\mathrm{R}^{CV}(\beta_{21}) = 0.113767$$

$$\mathrm{R}^{CV}(\beta_{22}) = 0.110999$$

We can get such the results

$$\mathrm{R}^{CV}(\beta_{21}) \leq \mathrm{R}^{CV}(\beta_{min}) + \mathrm{SE}^{CV}(\beta_{min})$$

$$\mathrm{R}^{CV}(\beta_{22}) \leq \mathrm{R}^{CV}(\beta_{min}) + \mathrm{SE}^{CV}(\beta_{min})$$

But for the $\mathrm{R}^{CV}(\beta_{20})$,

$$\mathrm{R}^{CV}(\beta_{20}) > \mathrm{R}^{CV}(\beta_{min}) + \mathrm{SE}^{CV}(\beta_{min})$$

That is, we can regard the 21th subtree as the optimal tree.

4.6 Optimal tree

The curve in Fig. 4.5 represents the performance of different sizes of trees on a test sample. In this case, the test sample was obtained by 10 fold cross validation which we discuss in Chapter 2. For now, we have an honest test base methodology for determining how well that given trees performs. The lower this curve gets the more accurate the model is because this is a measure of error.

According to 10 fold cross-validation and 1-SE rule, we can finally determine the 21th subtree as the optimal one.

When we set value cp ∈ [0.0025, 0.0029), in terms of α∈ [0.0005, 0.0006), we can get the optimal tree in R studio. The optimal tree has 92 splits which means it has 93 terminal nodes.

Use the follow formula, we can get the height of the optimal tree is

```
> nodesopt <- as.numeric(rownames(opttreeA$frame))

> max(rpart:::tree.depth(nodesopt))

[1] 19
```

That means 19 questions for clients. It is less than 28.

Now we describe the optimal tree performance evaluation. We use our training data `trainvaild` with 3973 observations. There are 3178 applicants who can pay the loan and 795 applicants who cannot pay the loan in truth. We create and describe the confusion matrix for prediction with optimal tree now. We classify 3359 applicant of them as being able to pay the loan and 614 applicants as not being able to pay the loan. The summary of this prediction can be seen in Tab 4.12.

Table 4.12 Optimal tree prediction of train data set

		Actual Values		
		0 (Good)	1 (Bad)	Total
Predicted	0 (Good)	3139 (79.01%)	220 (5.54%)	3359 (84.55%)
	1 (Bad)	39 (0.98%)	575 (14.47%)	614 (15.45%)
	Total	3178 (79.99%)	795 (20.01%)	3973 (100%)

The table includes cross-tabulation of predicted values of variable BAD and actual values of this variable. We can see that 3139+575 =3714 applicants were classified correctly. On the other hand, we have 220 applicants who cannot pay the loan but we classify them as being able and 39 applicants who are able to pay the loan and we classify them as bad applicants. The misclassification error of this classification is

$$R(T_{optimal}) = \frac{220 + 39}{3139 + 220 + 39 + 575} = 6.52\%$$

So the accuracy of this classification is 100% - 6.52%=93.48%.

4.7 Final evaluation

Now we start with the final performance evaluation shown in Fig. 2.2 We use our testing data `DfTest` with 1987 observations for this purpose. There are 1593 applicants who can pay the loan and 394 applicants who cannot pay the loan in truth. We create and describe the confusion matrix for prediction with optimal tree now. When we use the optimal tree model found in previous subchapter to predict whether they will be able to pay the loan or not we classify 1699 applicant of them as being able to pay the loan and 283 applicants as not being able to pay the loan. The summary of this prediction can be seen in Tab 4.13.

Table 4.13 Optimal tree prediction of test data set

		Actual Values		
		0 (Good)	1 (Bad)	Total
Predicted	0 (Good)	1524 (76.70%)	175 (8.81%)	1699 (85.81%)
Predicted	1 (Bad)	69 (3.47%)	219 (11.02%)	283 (14.49%)
	Total	1588 (80.17%)	394 (19.83%)	1987 (100%)

The table includes cross-tabulation of predicted values of variable BAD and actual values of this variable. We can see that 1524+219 =1743 applicants were classified correctly. On the other hand, we have 175 applicants who cannot pay the loan but we classify them as being able and 69 applicants who are able to pay the loan and we classify them as bad applicants. The misclassification error of this classification is

$$R(T_{optimal}) = \frac{175 + 64}{1524 + 175 + 64 + 219} = 12.06\%$$

So the accuracy of this classification is 100% - 12.06%=87.94%. To summarize, this model is able to recognize 219 of 394 bad clients from our testing data set, which is 55.58%, and cannot recognize 69 good clients out of 1593 which is 4.33%. We did not achieve as high accuracy as we achieved for the training data set which was 93.48%, but it's comparable to estimate of truth accuracy based upon cross-validation which was 10.95%.

5 Conclusion

Credit scoring is the set of decision models and their underlying techniques that aid lenders in the granting of consumer credit. These techniques determine who will get credit, how much scores they should get, and what operational strategies will improve the profitability of the borrowers to the lenders. There are only two actions possible—accept or reject—then there is no advantage in classifying this performance into more than two classes—good and bad. Good is any performance that is acceptable to the lending organization, while bad is performance that means the lender wishes he had rejected the applicant. That’s why the credit scoring is so important in reality.

In the thesis, we choose the Home Equity dataset which contains baseline and loan performance information for 5,960 recent home equity loans. Various machine learning techniques can be used to achieve a more accurate scoring from large data sets. We decided to use the CART algorithm to build a model to help us predict whether the applicant can pay the loan or not. The description of methods and algorithm in Chapter 2 and Chapter 3 aims to show how we use R studio for this purpose. Finally, we developed a model to help to automate the decision making process for home equity lines of credit. We used 2/3 of the data set to find the optimal model—for both training and validation--and 1/3 of the data set was used for the final performance evaluation.

The target binary variable (BAD) indicates whether and applicant eventually defaulted or was seriously delinquent (approximately 20% of cases) or not (approximately 80% of cases). With the help of optimal model developed in Chapter 4, we can expect our prediction to be correct with approximately 88% accuracy. One may consider such an improvement too low. There may have another better solution to improve. But as Thomas, Edelman and Crook (2002) said, “it is an area in which even a small improvement in performance can mean a tremendous increase in profit to the lender because of the volume of lending made by using scoring. Relevant, ubiquitous and profitable—credit scoring is all of these.”

Of course, there is still room for improvement and further questions may appear.

For example, we can take also more complicated loss function into account to distinguish type of misclassification, take uneven distributed classes into consideration or try to use ensemble methods to make the model more accurate.

Bibliography

[1] BREIMAN, L., J. H. FRIEDMAN, R. A. OLSHEN and C. J. STONE. *Classification and regression trees*.New York: Chapman & Hall, 1993. ISBN 978-0-412-04841-8.

[2] DALGAARD, Peter. *Introductory statistics with R*. New York: Springer, c2002. ISBN 0-387-95475-9.

[3] HAND, J. David and William, E. HENLEY. *Statistical Classification Methcxls in Consumer Credit Scoring: a Review.* Journal of the Royal Statistical Society Series A. 1997, Vol. 160, Issue 3, p. 523. ISSN 0964-1998.

[4] HANG LI. *Statistical Learning Method*. ISBN-13 978-7302275954

[5] THOMAS, L. C., David B. EDELMAN a Jonathan N. CROOK. Credit scoring and its applications. Philadelphia: Society for Industrial and Applied Mathematics, c2002. SIAM monographs on mathematical modeling and computation. ISBN 0-89871-483-4.

[6] Therneau, T.M. and Atkinson, E.J. (1997). *An Introduction to Recursive Partitioning Using the rpart Routines.* Technical Report 61, Section of Biostatistics, Mayo Clinic, Rochester. URL

[7] HILL, Thomas a Paweł LEWICKI. *Statistics: methods and applications*. Tulsa: StatSoft [Tulsa], c2006. ISBN 1-884233-59-7.

Electronic Bibliography

[8] https://www.r-project.org/

[9] https://www.rstudio.com/

[10] KAGGLE *HMEQ Data* [online] Available on https://www.kaggle.com/ajay1735/hmeq-data

[11] MEDIUM FREECODE CAMP. MFC *How to get a grip on Cross Validations* [online] MFC Available on https://medium.freecodecamp.org/how-to-get-a-grip-on-crossvalidations-bb0ba779e21c

List of abbreviations

CART	Classification and regression tree
MLR	Multiple linear regression
BLR	Binary logistic regression
LDA	Linear discriminant analysis
X_i	Independent variables
Y	Dependent variables
P	Probability
W_i	Weight
1 R method	One rule method
I	Impurity
R(T)	Misclassification error
cp	Complexity parameter
SE	Standard error
R^{CV}	Cross-validation error
$\lvert T \rvert$	Number of leaf nodes

Declaration of Utilisation of Results from a Bachelor Thesis

Herewith I declare that

- I am informed that Act No. 121/2000 Coll. – the Copyright Act, in particular, Section 35 –Utilisation of the Work as a Part of Civil and Religious Ceremonies, as a Part of School Performances and the Utilisation of a School Work – and Section 60 – School Work, fully applies to my bachelor thesis;
- I take account of the VSB – Technical University of Ostrava (hereinafter as VSB-TUO) having the right to utilize the bachelor thesis (under Section 35(3)) unprofitably and for own use;
- I agree that the diploma (bachelor) thesis shall be archived in the electronic form in VSB-TUO's Central Library. I agree that the bibliographic information about the diploma (bachelor) thesis shall be published in VSB-TUO's information system;
- It was agreed that, in case of VSB-TUO's interest, I shall enter into a license agreement with VSB-TUO, granting the authorization to utilize the work in the scope of Section 12(4) of the Copyright Act;
- It was agreed that I may utilize my work, the bachelor thesis or provide a license to utilize it only with the consent of VSB-TUO, which is entitled, in such a case, to claim an adequate contribution from me to cover the cost expended by VSB-TUO for producing the work (up to its real amount).

Ostrava dated 10. 5. 2019

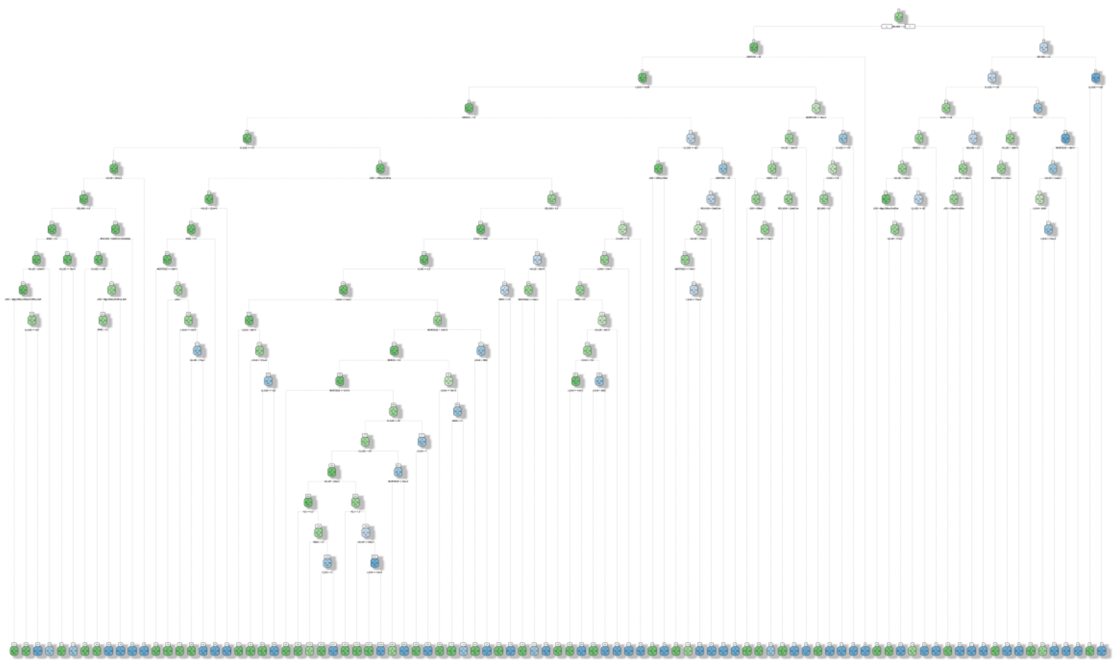
Songyang Gao 高松洋

....

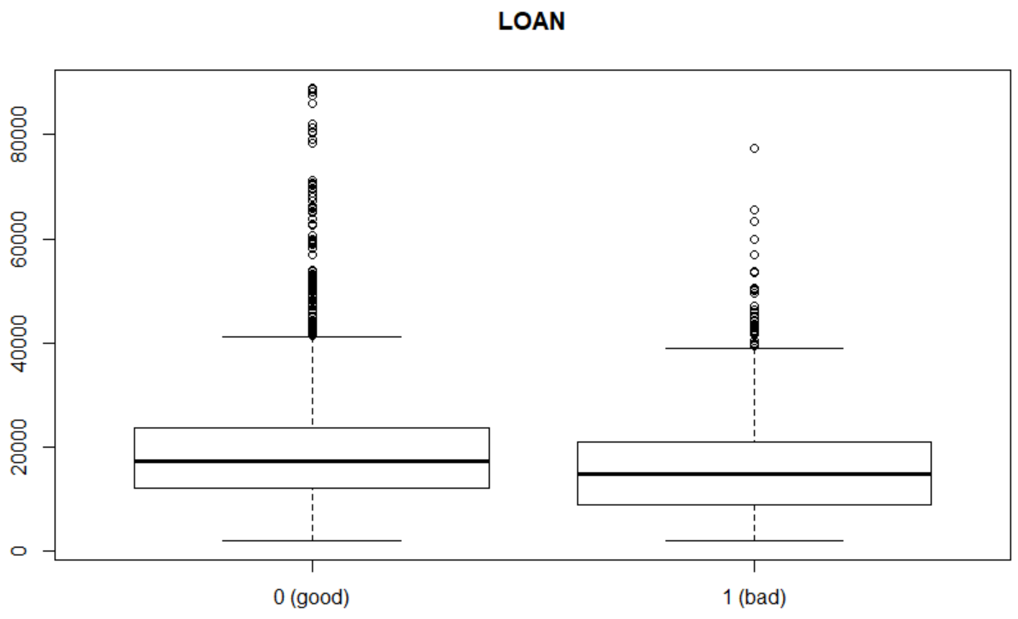
SONGYANG GAO

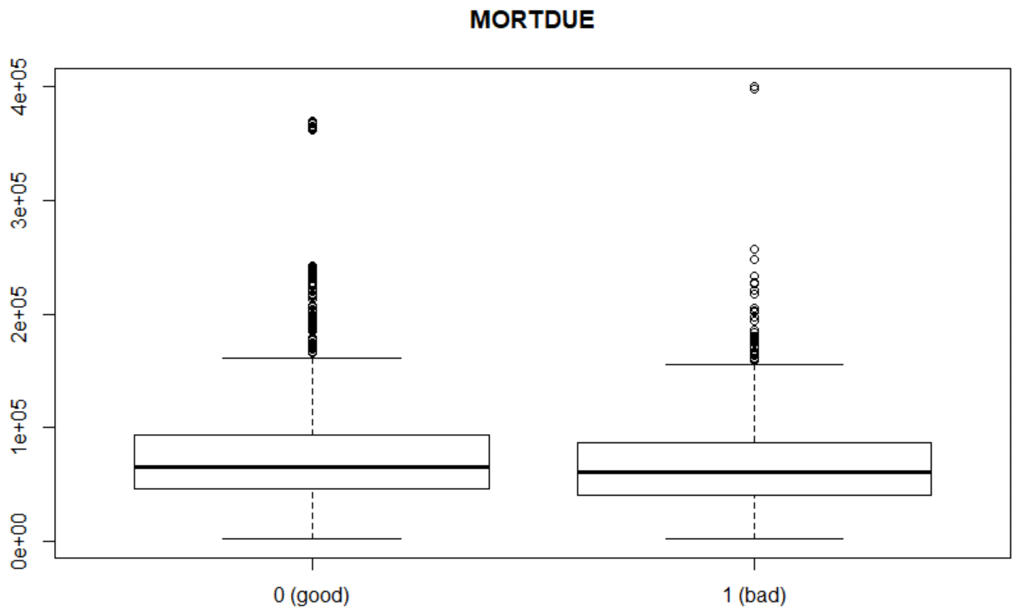
List of Annexes

Annex 1: The optimal tree

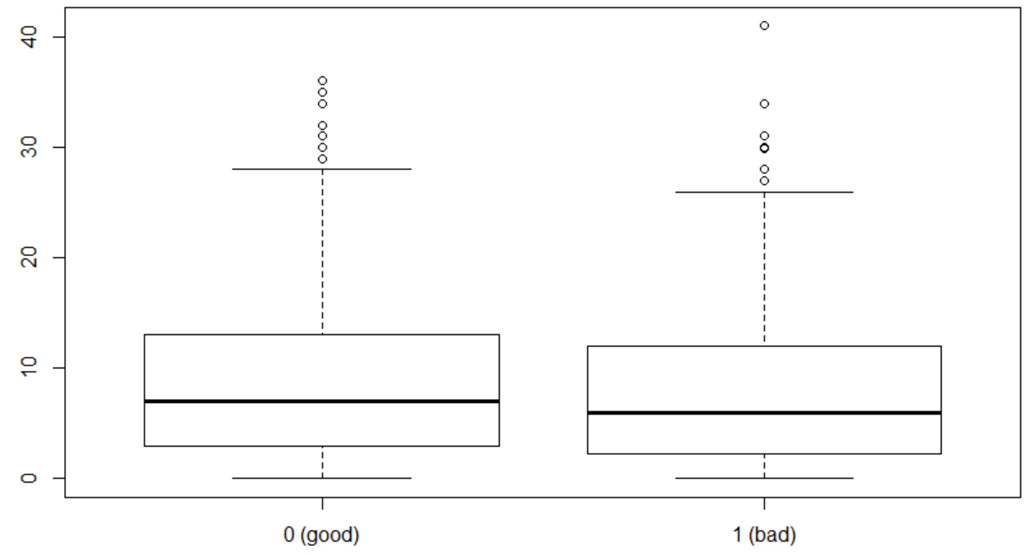


Annexes 2: The graphs of data

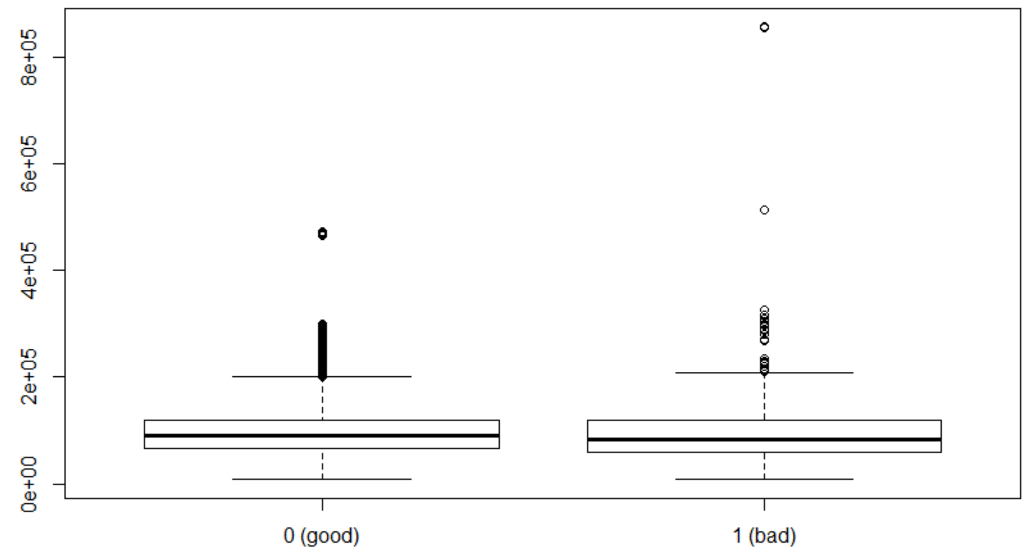




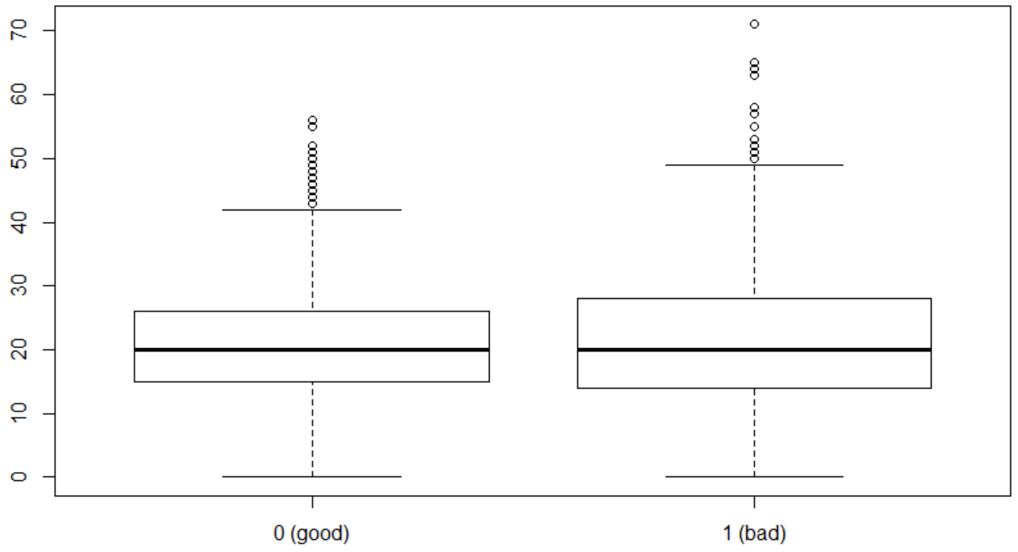
YOJ



VALUE

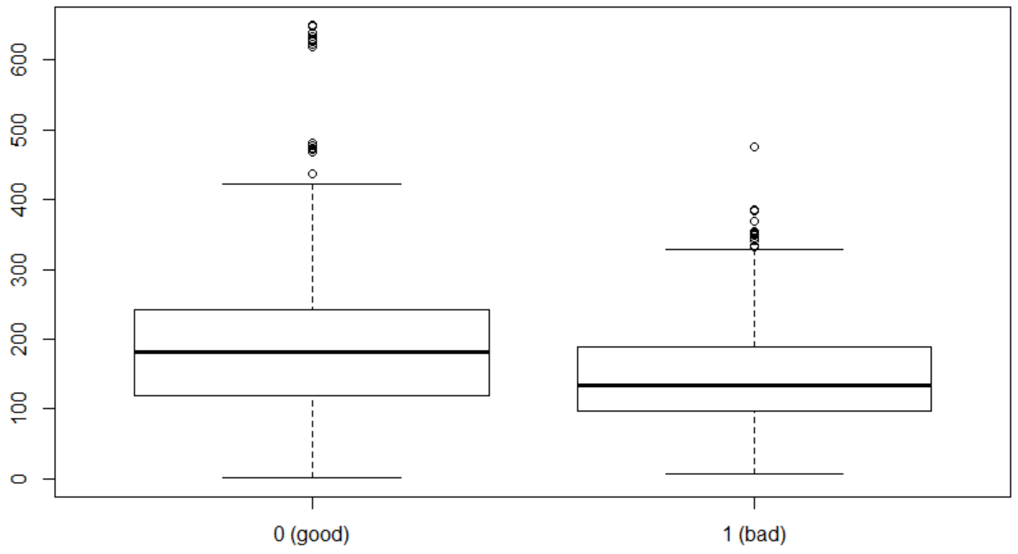


CLNO

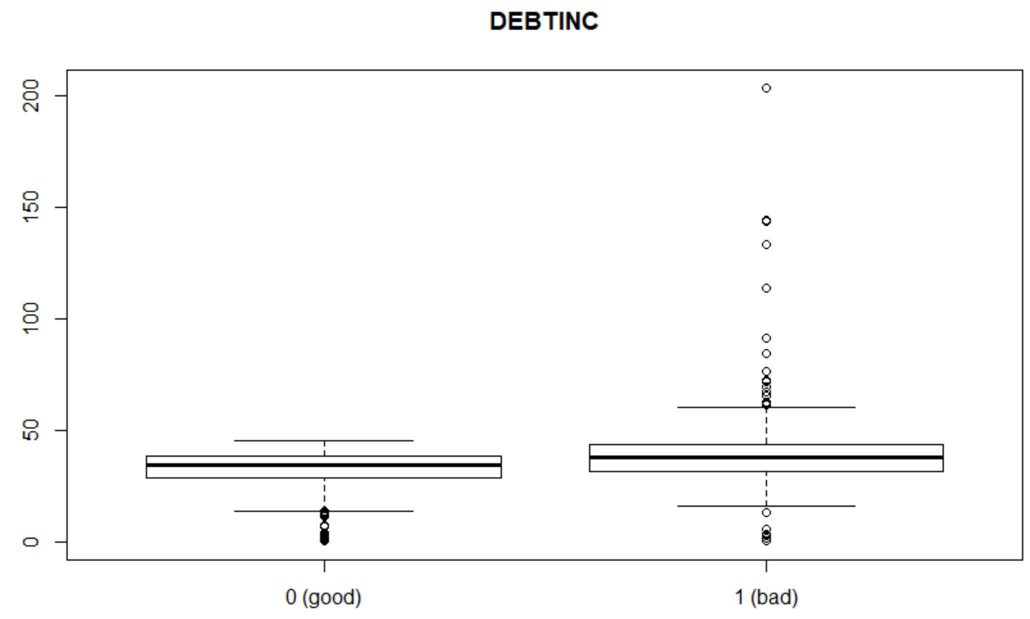


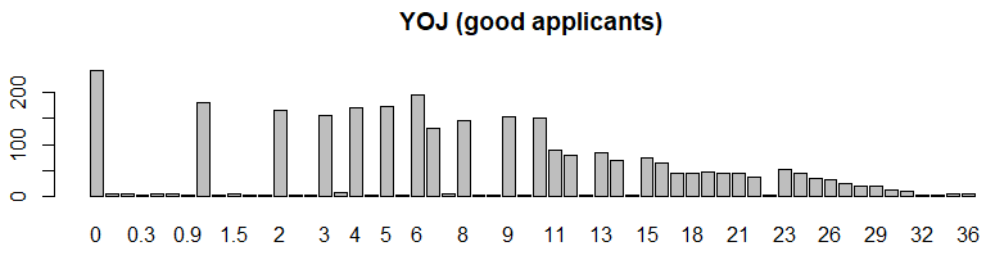
0
10
20
30
40
50
60
70
0 (good)
1 (bad)

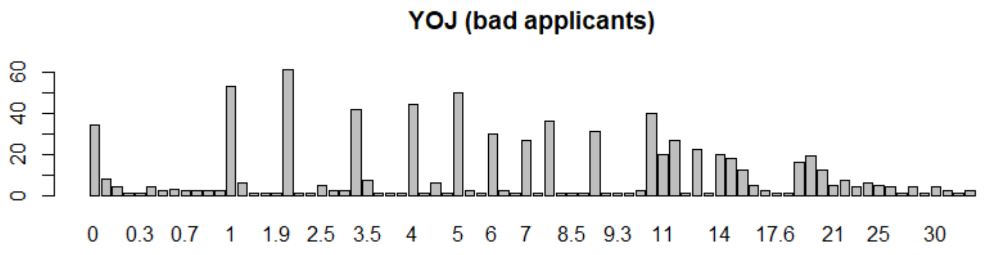
CLAGE

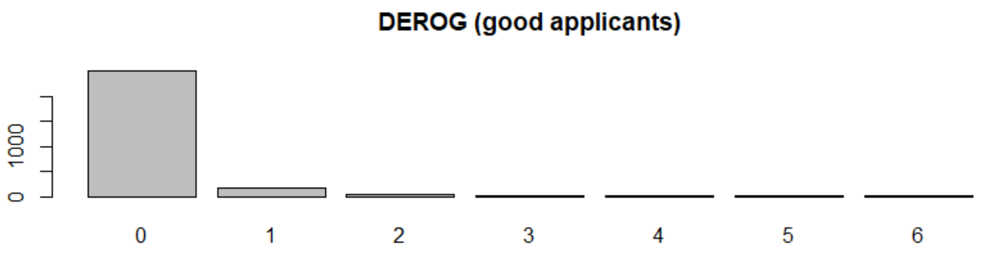


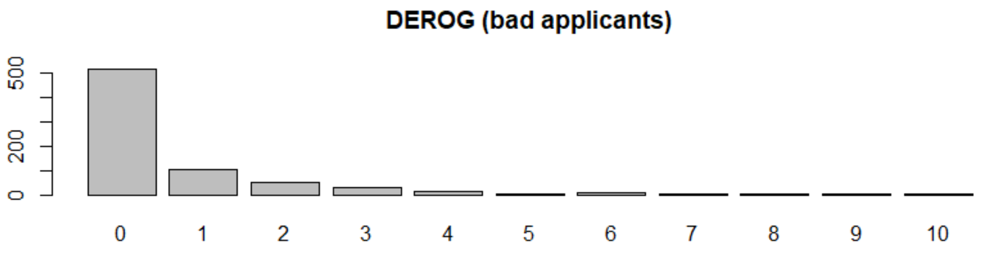
0
100
200
300
400
500
600
0 (good)
1 (bad)

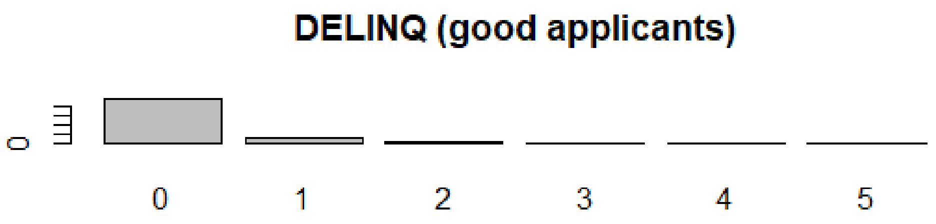

DEBTINC
0
50
100
150
200
0 (good)
1 (bad)

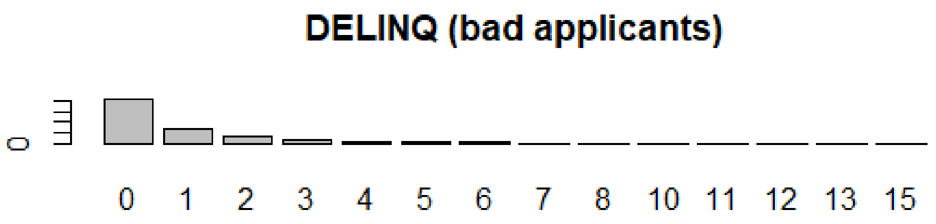

YOJ (good applicants)
0
100
200
0
0.3
0.9
1.5
2
3
4
5
6
8
9
11
13
15
18
21
23
26
29
32
36

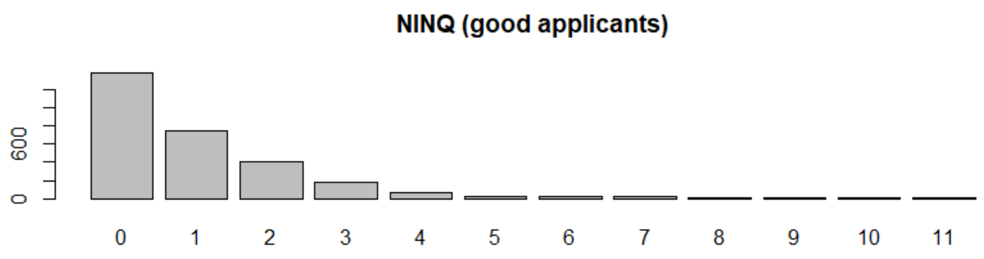

YOJ (bad applicants)
0
20
40
60
0
0.3
0.7
1
1.9
2.5
3.5
4
5
6
7
8.5
9.3
11
14
17.6
21
25
30

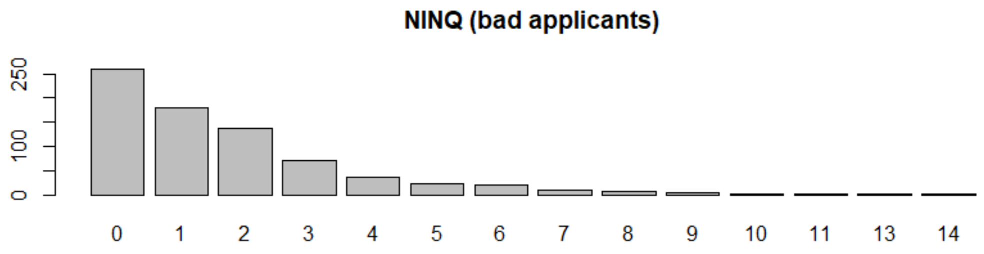

DEROG (good applicants)
0
1000
0 1 2 3 4 5 6

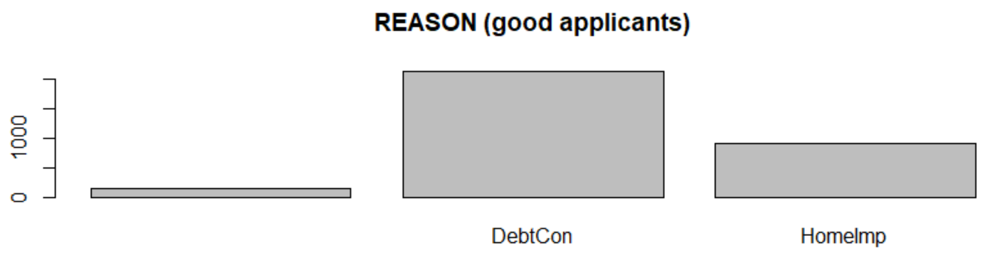

DEROG (bad applicants)
0
200
500
0 1 2 3 4 5 6 7 8 9 10

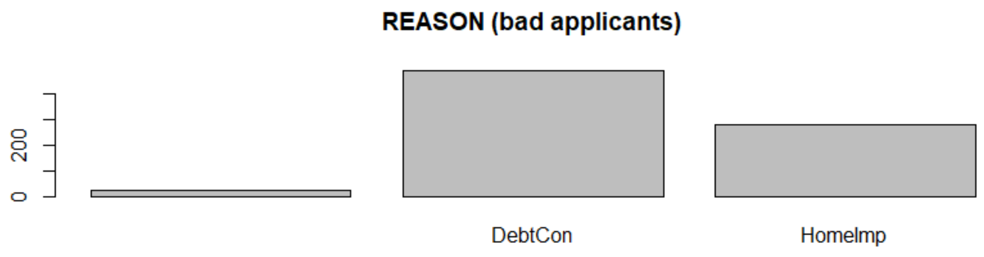

DELINQ (good applicants)
0
0 1 2 3 4 5


DELINQ (bad applicants)
0
0 1 2 3 4 5 6 7 8 10 11 12 13 15


NINQ (good applicants)
0
600
0
1
2
3
4
5
6
7
8
9
10
11


NINQ (bad applicants)
0
100
250
0
1
2
3
4
5
6
7
8
9
10
11
13
14


REASON (good applicants)
0
1000
DebtCon
HomeImp


REASON (bad applicants)
0
200
DebtCon
HomeImp

Annexes 3: The whole cp table generated by `printcp`

	CP	nsplit	rel error	xerror	xstd
1	0.07421384	0	1.0000000	1.0000000	0.031720
2	0.05660377	1	0.9257862	0.9710692	0.031371
3	0.02830189	2	0.8691824	0.9018868	0.030491
4	0.02327044	4	0.8125786	0.8691824	0.030053
5	0.01509434	6	0.7660377	0.8226415	0.029401
6	0.01006289	9	0.7207547	0.7786164	0.028754
7	0.00880503	11	0.7006289	0.7773585	0.028735
8	0.00754717	12	0.6918239	0.7572327	0.028428
9	0.00712788	13	0.6842767	0.7459119	0.028253
10	0.00628931	16	0.6628931	0.7371069	0.028114
11	0.00587002	20	0.6377358	0.7333333	0.028055
12	0.00566038	32	0.5647799	0.7257862	0.027935
13	0.00503145	34	0.5534591	0.7106918	0.027692
14	0.00440252	40	0.5232704	0.6905660	0.027361
15	0.00431267	48	0.4867925	0.6729560	0.027065
16	0.00408805	55	0.4566038	0.6729560	0.027065
17	0.00377358	59	0.4402516	0.6339623	0.026387
18	0.00314465	71	0.3949686	0.6138365	0.026025
19	0.00293501	74	0.3849057	0.6188679	0.026116
20	0.00287511	82	0.3572327	0.6188679	0.026116
21	0.00251572	92	0.3257862	0.5685535	0.025175
22	0.00188679	110	0.2805031	0.5547170	0.024906
23	0.00167715	132	0.2364780	0.5471698	0.024757
24	0.00125786	135	0.2314465	0.5534591	0.024881
25	0.00107817	232	0.1081761	0.5522013	0.024856
26	0.00104822	242	0.0968553	0.5522013	0.024856
27	0.00094340	249	0.0893082	0.5547170	0.024906
28	0.00083857	258	0.0805031	0.5597484	0.025004
29	0.00075472	273	0.0679245	0.5660377	0.025127
30	0.00062893	288	0.0553459	0.5836478	0.025464
31	0.00055905	352	0.0150943	0.5911950	0.025606
32	0.00050314	363	0.0088050	0.5911950	0.025606
33	0.00041929	372	0.0037736	0.5962264	0.025700
34	0.00000000	380	0.0000000	0.5962264	0.025700