UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

# Multivariate Analysis

Exercise 3

**Students:** Denaldo Lapi, Francesco Aristei, Samy Chouiti

# Contents



# List of Figures

# List of Tables

# 1 Introduction

The goal of this report is to describe our solutions and interpretations of the two problems of Exercise 3 of the Multivariate Analysis course, solved by using the R software.

The problems are related to the following methods:

- Linear Discriminant Analysis (LDA) and its extensions
- Classification Trees and Random Forests

## 1.1 R scripts and packages

We built 2 R notebooks (*.Rmd* files), one per each problem, in which we describe in details all the steps of our analysis. For each problem we follow the same notebook structure:

- Data exploration and preprocessing: explore and apply some preprocessing steps to the loaded data in order to make it compatible for the problem to solve and in order to improve the quality of the dataset
- Execution of the multivariate analysis method: we apply the algorithm of interest among the ones mentioned before
- Results interpretation: describe the results obtained after applying the algorithm

## 1.2 Some notes about the report

We would like to remark that the 2 provided notebooks contain a detailed analysis of each one of the problems, with all the steps that we followed to execute and interpret each analyzed method. In this report we'll describe only the most meaningful steps and results we obtained.
We also added an appendix in the end of the report containing the most important plots, so that this document could be able to provide a self-contained analysis of the 2 problems. However, we strongly suggest to read the R notebooks for a more detailed and clear explanation of all the solutions.

# 2 Linear Discriminant Analysis (LDA) and its extensions

The first problem required us to apply the *Linear Discriminant Analysis method* and its extensions *QDA, FDA, RDA, MDA*. The main goal of the assignment was to prepare the dataset for applying the various methods and then compare their performances.

## 2.1 Dataset description

We were provided with the *phoneme.csv* dataset, which contains samples of digitized speech for five phonemes: 'aa' (as the vowel in dark), 'ao' (as the first vowel in water), 'dcl' (as in dark), 'iy' (as the vowel in she), and 'sh'(as in she). In total, 4509 speech frames of 32 msec were selected. For each speech frame, a log-periodogram of length 256 was computed, on whose basis we want to perform speech recognition. The 256 columns labeled x.1 to x.256 identify the speech features, while the columns g and speaker indicate the phonemes (labels) and speakers, respectively. We performed the various methods by taking into account the first 10 columns, i.e., from x.1 to x.10, and the labels (column g).

## 2.2 Exploratory data analysis

After loading the dataset, we performed some exploration over it, by visualizing the summary and by selecting only the columns of interests. We then converted the variables into the correct type for our analysis.

As a following step, we checked for missing values and we performed some multivariate outlier detection.

### 2.2.1 Outlier detection

For what regards outlier detection, we tried 2 different approaches:

- *Malhanobis' distance*[1]
- *Local outlier factor (LOF)*[2]

The *Malhanobis' distance* can be thought of as a metric for estimating how far each observation is from the center of all the variables' distributions (i.e. the centroid in the multivariate space). We used the *chemometrics* R package, which allows to calculate the distances for all the observations (by providing the quantile cutoff point beyond which points should be treated as outliers ) and which returns a cutoff value for outliers. Then, by simply considering observations with a distance value above the cutoff, we obtained the indexes of the outliers.

The *LOF* algorithm is instead a non-parametric approach. Basically, it computes a local density for observations with a given k-nearest neighbors (we chose k = 5). This local density is compared to the density of the respective nearest neighbors, resulting in the local outlier factor. We used the *DDoutlier* package for applying the algorithm, which return a vector of LOF scores for each observation: the greater the LOF, the greater the outlierness of the data point. The considered package allows to visualize the distribution of outlier scores (figure 3), which suggested us that the poins to be considered as outliers are the ones with LOF score above 1.4. By using this cutoff value, we simply selected the indexes of the elements with an higher LOF score.

Finally, we simply decide to remove the found outliers from our dataset, since their number (172) is pretty small if compared to the dataset size (4509).

### 2.2.2 Check predictors normal distribution

We proceeded our exploration by analyzing the univariate distributions of our numeric predictors. At first, we simply applied a visual inspection by checking the density plots and the *Q-Q plots* for each variable, which are shown in the figure 4. In particular, by analysing the last ones, which show the distribution of the data against the expected normal distribution, we concluded that none of the predictors follows a normal distribution, except for the first one ('x.1'). In order to get some more precise insights, we performed the *Shapiro-Wilk normality test*[3] which returned very low p-values for each variable, therefore rejecting the null hypotheses about the normal distribution.

In order to obtain some improvements on the predictors distributions and to make them follow a bell shape, we tried the *BoxCox*[4] transformations, by using the package *forecast*. The

[1] Malhanobis' distance
[2] LOF
[3] Shapiro-Wilk normality test
[4] Box-Cox paper

*BoxCox* method applies a power transformation, which depends on a parameter . The function 'BoxCox.lambda' of the package finds the optimal value of the parameter by maximizing the profile log likelihood of a linear model fitted to our data (by specifying the value 'loglik' in the 'method' parameter) . We applied those transformation over all the variables and we checked the results by using again the *Shapiro-Wilk test*, which showed that only the predictor 'x.1' was following a normal distribution.

#### 2.2.3 Classes distributions

Being LDA a supervised dimensionality reduction technique, we decided to analyze the distributions of the values for each predictor broken down by each class, since our dataset contains labels for each class in the 'g' variable. We therefore plotted the *boxplots* and the *densities*. In particular, these last ones, reported in the figure 5, showed the differences among the 5 classes: for instance 'sh' is the phoneme with the lowest values of the features, except for x.1, x.2 and x.10; while 'iy' is on average the phoneme with highest values.

We then plotted the correlation matrix to check the correlation among the numerical variables, which showed that they are actually pretty correlated.

#### 2.2.4 Scaling

Discriminant analysis is not affected by the scale/unit in which predictor variables are measured, but still we decided to standardize the variables to make their scale comparable: we therefore applied a scaling to our predictors so that they assume zero mean and unit variance.

#### 2.2.5 Data splitting

In order to evaluate the performance of the discriminant analysis methods, we decided to split the dataset into 2 subsets, i.e. we use 80% of the initial dataset as training data and the remaining 20% as test data. In particular, we fitted our discriminant analysis models on the train datasets and we evaluated them on the test dataset, in order to obtain not biased results. We performed this operation by using the *caret* R library, which allows to perform an homogeneous splitting, i.e. it splits the data by mantaining the same distribution among the vairious classes: this ensures to have test data where the class distribution is the same as the one of the train data and this allows to have more reliable performance evaluations.

### 2.3 Running LDA and its extensions

At that point, we were able to apply the LDA method and his extensions on the previously analyzed data. We run the various methods by following the same steps for each one of them:

- Run the method over the training set
- Compute predictions over the test set
- Compute accuracy over the test set

The R packages we used for applying the methods are:

- *MASS* package for performing LDA and QDA
- *mda* package for per FDA and MDA
- *klaR* package for RDA

In particular, we would like to remark that for each performed discriminant analysis method we printed a table (through kable) showing the posterior probabilities from the test data for the first 6 rows. The resulting tables for each method can be shown in the figure 6.

The detailed description of the performed steps of each algorithm are described inside the provided notebook *LDA.rmd* with all the interpretations and results discussions.

### 2.4 Conclusions

The analyses we performed showed that there is almost no difference among the various techniques we used for applying *discriminant analysis* on the "phoneme" dataset. Indeed all the models obtained almost the same accuracy value over the test dataset, around 78%.

In particular, we obtained our best result with the QDA model, which correctly classified 78.64% of the test observations ; therefore we may say that, for this dataset, the spread in each class is different and QDA takes advantage of that, allowing to obtain a (very small) improvement w.r.t. LDA and the other considered extensions.

## 3 CART and Random Forest

In this section, we'll apply Classification Tree and Random Forest algorithms to the *Ionosphere*[5] from the mlbench library.

### 3.1 Dataset exploration

According to its documentation, this data set include "[...] radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. [...] The targets were free electrons in the ionosphere. "good" radar returns are those showing evidence of some type of structure in the ionosphere. "bad" returns are those that do not; their signals pass through the ionosphere." [6].

The dataset contains 35 variables:

- Class variable: binary categorical variable stating whether a measurement is good or bad, which we will try to predict
- 34 "V" variables: numerical variables corresponding to the 17 probes measures (a real and complex measure per probe).

From an initial exploration, we can see that the dataset contains almost twice observations being "good" (225) as observation being "bad" (126). Therefore, we'll have to be careful of this characteristic when drawing conclusions about our models as we could be better at predicting "good" classes than "bad" ones.

Interestingly, when *V3* to *V34* variables contains normalized numerical values between -1 and 1 [7], *V1* is only taking 0 or 1 values and *V2* only the 0 value, thus making it useless as not

[5] https://archive.ics.uci.edu/ml/datasets/ionosphere
[6] Source : https://www.rdocumentation.org/packages/mlbench/versions/2.1-3/topics/Ionosphere
[7] This normalization step has been done by the original papers authors.

changing for observations. [8]

## 3.2 Train/test split

**Answer a)** To build our model, we'll do a random 80/20 train/test split on our dataset. Because we want to keep the same "good"/"bad" distribution in test and train datasets as in the initial one, we will use the *createDataPartition* method from the *caret* package to do so.

The table 1 containing the *Class* variable distribution ensures us that we will proceed the application of Classification Trees and Random Forest models on sets with good data representation.

| Set origin | "bad" | "good" |
|---|---|---|
| Initial set | 126 | 225 |
| Train set | 100 | 181 |
| Test set | 26 | 44 |

Table 1: Class distribution in different sets

## 3.3 Classification tree

As our first model, we will apply a simple classification tree on the previously train/test set with different Complex Parameter (CP) values: 0.3, 0.2, 0.1 and 0.05. We will then plot their different classification trees and describe them. Then we will determine the optimal CP value and analyse its performances.

### 3.3.1 First iterations

**Answer b and c)** We computed Classification Trees with several Complexity Parameters values and obtained different classification trees 7 8 9 and 10.

We can observe several similarities between those Classification Trees (CF):

- For CP = 0.3, 0.2 or 0.1, the V5 variable is used as a first criterion (for CP=0.05, V6 is used instead)
- Except for CP=0.3 (as having only one split), all CF's uses V27 as second split criterion

As expected, the smaller the CP value is, the more split and leaves we have 2.

| CP Value | Leaves | Splits |
|---|---|---|
| 0.3 | 2 | 1 |
| 0.2 | 3 | 2 |
| 0.1 | 3 | 2 |
| 0.05 | 5 | 6 |

Table 2: Split and Leaves number per CP value

[8] Question b) asks us to still use the 34 predictors to fit the class variable, thus we will use all the features for the rest of the assignment, including *V2*.

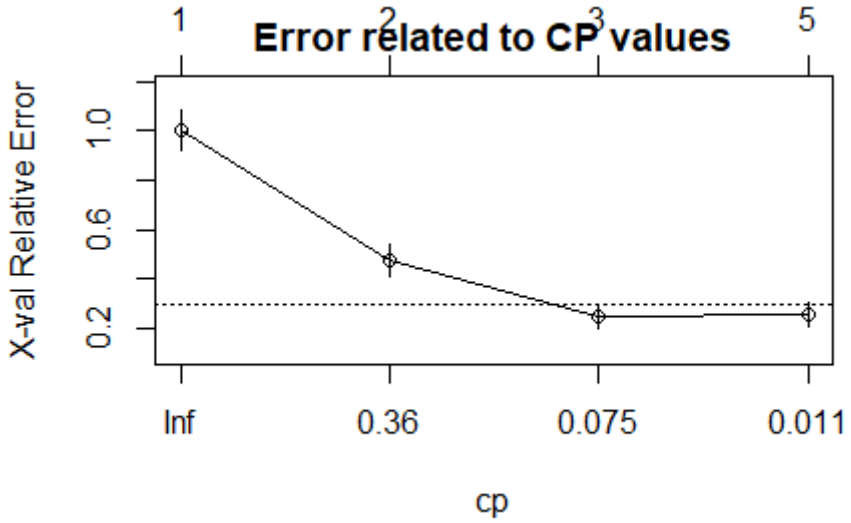



Figure 1: Error related to CP values

#### 3.3.2 Performances on an optimal Complexity Parameter

**Answer d)**After gathering the above Classification Tree, we would like to know if there is a more optimal CP could be chosen 1 in order to improve the performances of our model. To so, we will compute a last Classification Tree using the optimal CP and use a **simple accuracy** and as there is a consequent disparity in the distribution of "good"/"bad" observation, we'll also compute the **balanced accuracy score**, which will take into account the data distribution.

Using the optimal CP parameter 0.025, we then computed a simple accuracy score of 83% and a balanced accuracy of 80% when we had only respectively 79% and 73% with cp=0.05, thus introducing significant improvements on both metrics.

The balanced accuracy being lower than the simple accuracy mean that the least represented class (which is "bad" here) is having a stronger weight (in fact, as weighted as the other class) in the accuracy score.

Thus, we can deduce that predictions are better on *class="good"* observation than on *class="bad"*.

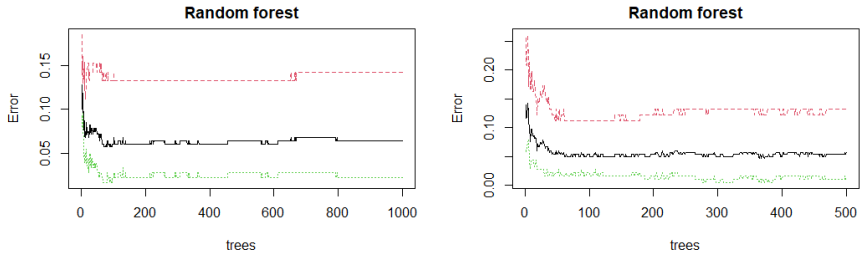
Finally, we can confirm this conclusion with the confusion matrix 3 showing that only 2 observations were misclassified as good (when supposed to be bad) when 5 observations were misclassified as bad (when supposed to be good).

| | bad | good |
|---|---|---|
| bad | 21 | 2 |
| good | 5 | 42 |

Table 3: Confusion Matrix

### 3.4 Random Forest

**Answer e)** In this section, we will use a Random Forest to obtain better performances on this dataset classification. Also, to improve performances, we will first get the most optimal number




(a) General overview (b) Zoom on the 0 - 500 range

Figure 2: Tree number effects on error (black="bad", green="good", red=total)

of trees then number of parameters to take into account in the Random Forests computation. As a comparison, we will run first the Random Forest model with default parameter.

**Initial simple accuracy: 90% and balanced accuracy: 87%**

### 3.4.1 Number of variables

As a first optimization of our model, will try to find the optimal number of variable used in our trees. To do so, we iterated over a range of variable number and plot the OOB error rate 11.

The OOB error seems to be reaching a global minimum around 16 variables.

**After number of variable optimization: simple accuracy: 90% and balanced accuracy: 87%**

### 3.4.2 Number of trees

Variable number of trees 2 can heavily improve the prediction quality of our model.

In this case, we can see that the error rates are not improving after n=130.

**After tree number optimization: simple accuracy: 88.6% and balanced accuracy: 86,7%**

Interestingly, our accuracy is not improving but rather decreasing with this optimization. We can interpret this as a better fitting of the model to the test dataset with former parameters of the tree number, as the plot that helped us choosing this number is based on train data errors (and not test).

### 3.4.3 Going further with feature engineering

From a purely physical point of view, we could also take into account the norm of the electromagnetic field into consideration to state whether a signal is "good" or "bad". Therefore, we'll add the norm of each couple of measurement (real and complex values) and study its influence on the Random Forest performances.

As did before, we applied a balanced train/test split of 80/20 then, optimized the number of variables used then the number of trees. With ntrees=100 and mtry=6, we had performances of **simple accuracy: 95.7% and balanced accuracy: 95,8%**, which is significantly better

than before.

Most important features according to the random forest computation were (decreasing order): N13[9], V5, V3 and N16. Therefore, the norm of the complex number is a relevant feature for this classification problem.

### 3.5 Conclusions

In this section, we saw the application of Classification Trees and Random Forest to this dataset. Through a series of different optimization on those model's parameter, we have been able to increasingly improve our model to correctly classify given measurements.

Although having interesting results with the Classification Tree model, we saw that Random Forest - even with default paramters - were performing better 4.

Besides that, we showed a simple example of feature engineering by computing the norm of the complex values to add features to the dataset, in the hope for better predictions. We saw that it was relevant as we went from 90% to 95,8% on simple accuracy when adding the computed norms to the dataset. Other types of feature engineering could have been done by taking the physical nature of the dataset.

All the results of our analysis have been gathered here:

| | Simple | Balanced |
|---|---|---|
| CT, default | 79% | 73% |
| CT, optimal CP | 83% | 80% |
| RF, default | 90% | 87% |
| RF, optimal mtry | 90% | 87% |
| RF, optimal ntree | 88,60% | 86,70% |
| **RF, optimal mtry + feature eng.** | **95,7%** | **95,80%** |

Table 4: Compilation of obtained accuracy scores (CT: Classification Tree, RF: Random Forest)

[9] norm of the 13th probe measurements, computed by ourselves

# 4 Appendix

## 4.1 LDA and extensions

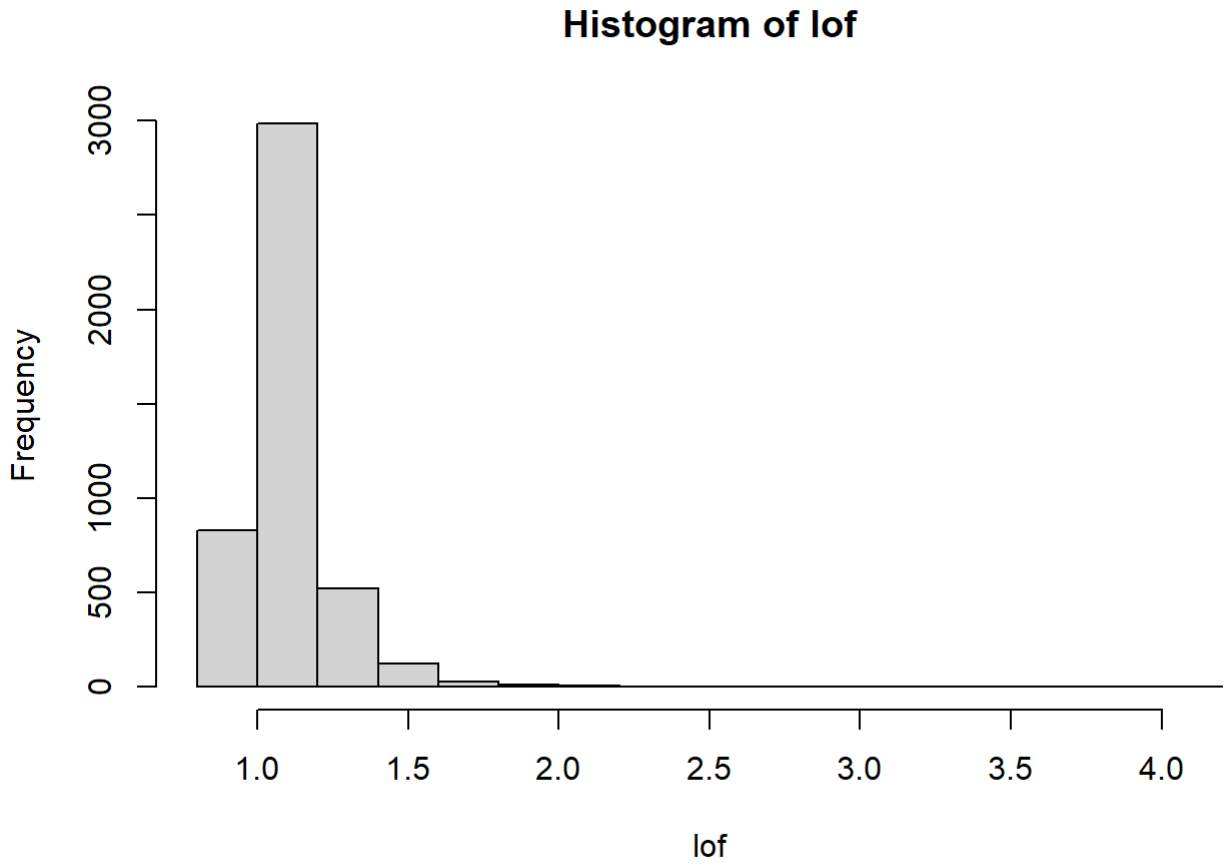



Figure 3: Histogram of outlier scores

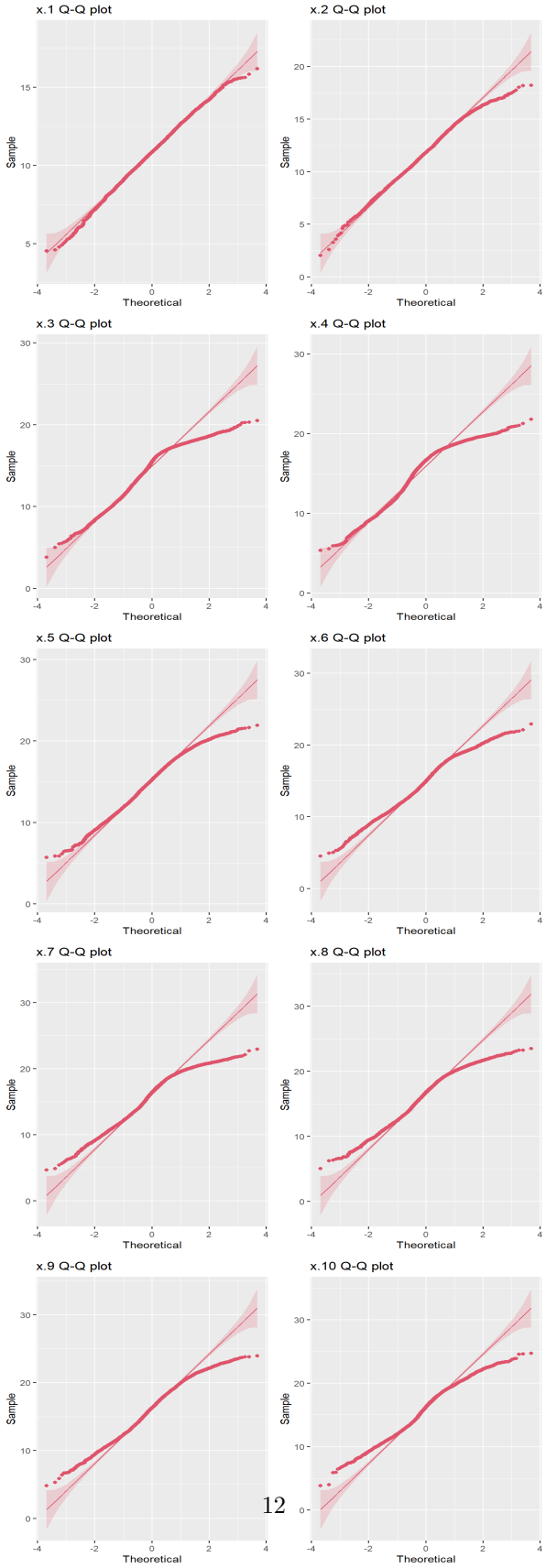





Figure 4: Q-Q plots

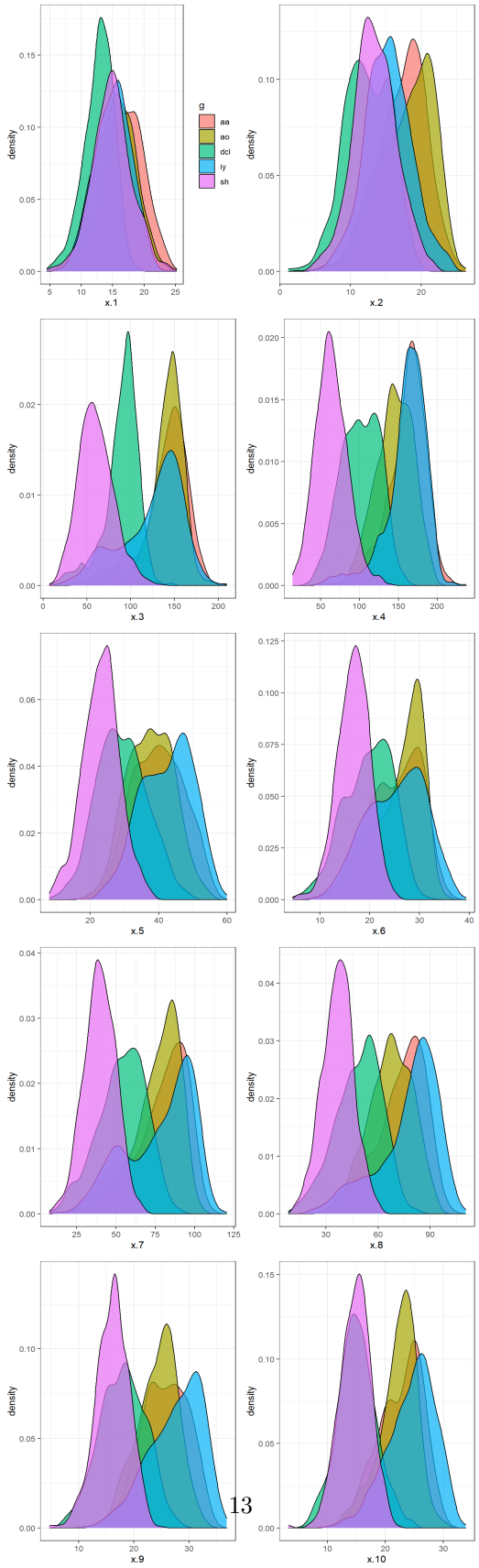





Figure 5: Distribution of the values for each attribute broken down by each class

Predicted probabilities of class membership (first 6 rows)

| | aa | ao | dcl | iy | sh |
|---|---|---|---|---|---|
| 9 | 0.2705627 | 0.6772213 | 0.0045311 | 0.0476849 | 0.0000000 |
| 11 | 0.0000000 | 0.0000000 | 0.0016722 | 0.0000000 | 0.9983278 |
| 12 | 0.4572327 | 0.4352354 | 0.0000001 | 0.1075319 | 0.0000000 |
| 16 | 0.3824746 | 0.5435839 | 0.0000022 | 0.0739393 | 0.0000000 |
| 27 | 0.0000000 | 0.0000000 | 0.0001106 | 0.0000000 | 0.9998894 |
| 40 | 0.0027246 | 0.0055984 | 0.9900036 | 0.0013774 | 0.0002960 |

Predicted probabilities of class membership (first 6 rows)

| | aa | ao | dcl | iy | sh |
|---|---|---|---|---|---|
| 9 | 0.1048399 | 0.8921249 | 0.0000681 | 0.0029671 | 0.0000000 |
| 11 | 0.0000000 | 0.0000000 | 0.0174347 | 0.0000000 | 0.9825653 |
| 12 | 0.5730533 | 0.2670043 | 0.0000000 | 0.1599424 | 0.0000000 |
| 16 | 0.1798585 | 0.8119907 | 0.0000000 | 0.0081508 | 0.0000000 |
| 27 | 0.0000000 | 0.0000000 | 0.0019285 | 0.0000000 | 0.9980715 |
| 40 | 0.0002650 | 0.0000001 | 0.9979270 | 0.0018079 | 0.0000000 |

Predicted probabilities of class membership (first 6 rows)

| | aa | ao | dcl | iy | sh |
|---|---|---|---|---|---|
| 9 | 0.2704687 | 0.6775138 | 0.0045024 | 0.0475152 | 0.0000000 |
| 11 | 0.0000000 | 0.0000000 | 0.0016572 | 0.0000000 | 0.9983428 |
| 12 | 0.4574929 | 0.4352151 | 0.0000001 | 0.1072920 | 0.0000000 |
| 16 | 0.3825695 | 0.5436985 | 0.0000022 | 0.0737298 | 0.0000000 |
| 27 | 0.0000000 | 0.0000000 | 0.0001092 | 0.0000000 | 0.9998908 |
| 40 | 0.0027020 | 0.0055548 | 0.9900870 | 0.0013637 | 0.0002925 |

Predicted probabilities of class membership (first 6 rows)

| aa | ao | dcl | iy | sh |
|---|---|---|---|---|
| 0.2675627 | 0.6810396 | 0.0044756 | 0.0469221 | 0.0000000 |
| 0.0000000 | 0.0000000 | 0.0017679 | 0.0000000 | 0.9982321 |
| 0.4565407 | 0.4353395 | 0.0000001 | 0.1081197 | 0.0000000 |
| 0.3786215 | 0.5493036 | 0.0000021 | 0.0720727 | 0.0000000 |
| 0.0000000 | 0.0000000 | 0.0001214 | 0.0000000 | 0.9998786 |
| 0.0026335 | 0.0052462 | 0.9903811 | 0.0014395 | 0.0002998 |

Predicted probabilities of class membership (first 6 rows)

| aa | ao | dcl | iy | sh |
|---|---|---|---|---|
| 0.3784877 | 0.5168569 | 0.0006743 | 0.1039798 | 0.0000014 |
| 0.0000000 | 0.0000000 | 0.0073780 | 0.0000000 | 0.9926220 |
| 0.3819473 | 0.5033004 | 0.0000000 | 0.1147523 | 0.0000000 |
| 0.2676083 | 0.6959388 | 0.0000000 | 0.0364528 | 0.0000000 |
| 0.0000000 | 0.0000000 | 0.0000333 | 0.0000000 | 0.9999667 |
| 0.0000144 | 0.0000001 | 0.9978496 | 0.0021359 | 0.0000000 |

Figure 6: Posterior probabilities from the test data set for the first 6 rows. In order from left to right and from top to bottom: LDA, QDA, FDA, RDA, MDA

### 4.2 Classification Trees and Random Forests

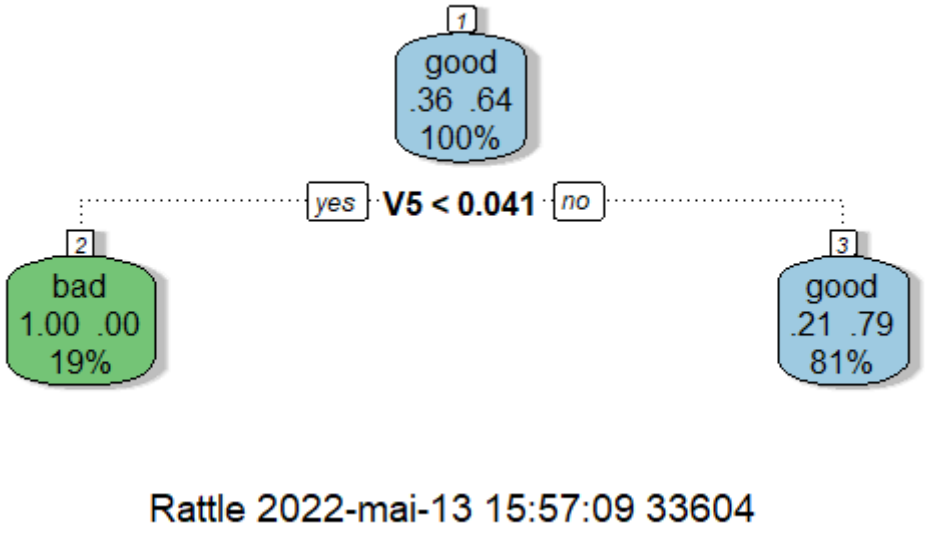



Figure 7: CP=0.3

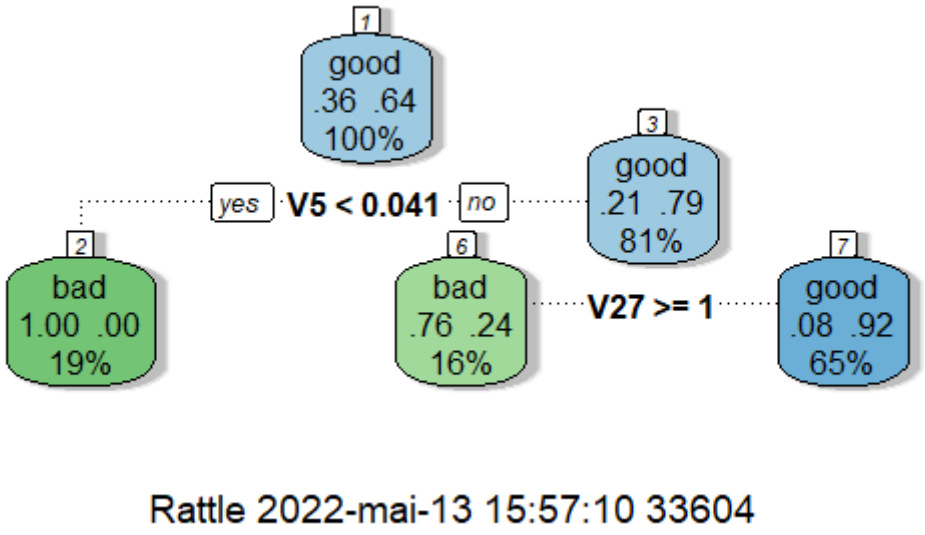



Figure 8: CP=0.2

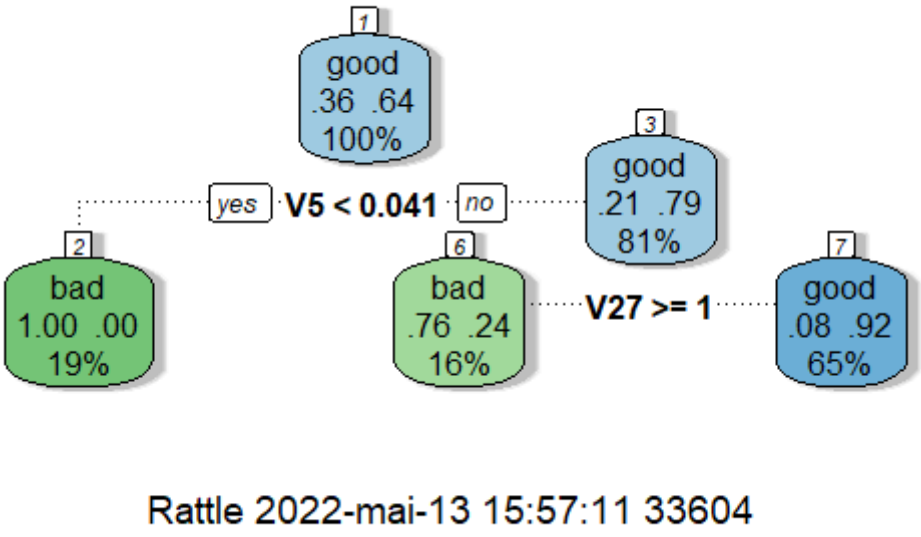



Figure 9: CP=0.1

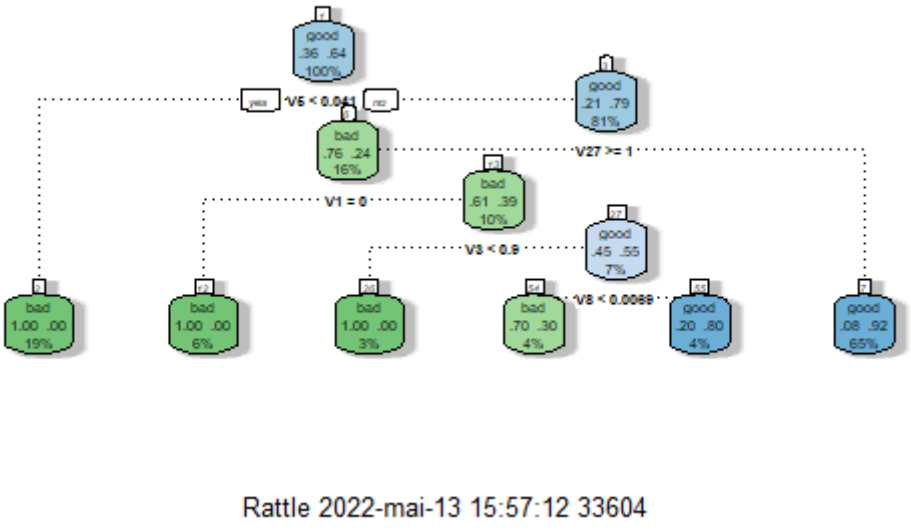



Figure 10: CP=0.05

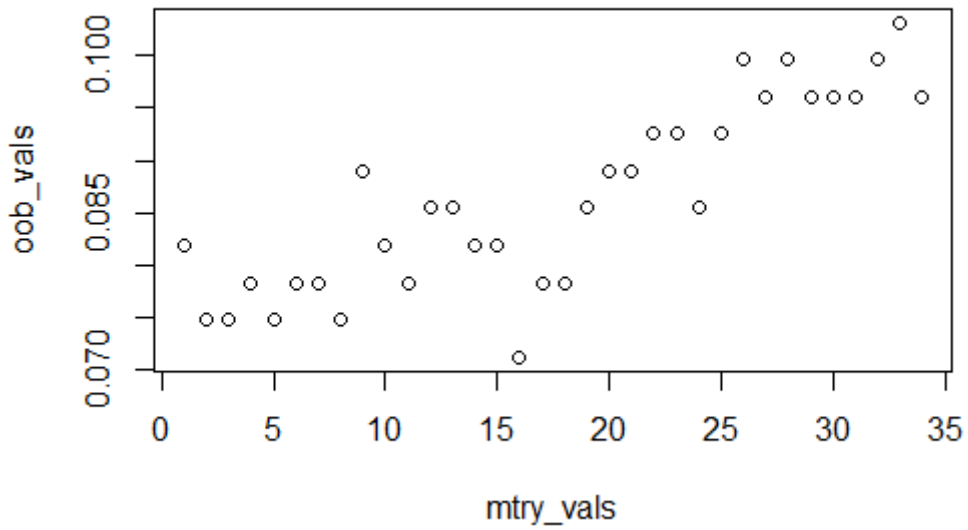



Figure 11: Out-of-box error for different number of variables (mtry)