Capstone2: German Credit Risk

Louis Mono

May, 2019

Contents

Abstract

A **credit score** is a numerical expression based on a level analysis of a person's credit files, to represent the creditworthiness of an individual. Lenders, such as banks and credit card companies, use credit scores to evaluate the potential risk posed by lending money to consumers and to mitigate losses due to bad debt. Generally, financial institutions utilize it as a method to support the decision-making about credit applications. Both Machine learning (ML) and Artificial intelligence (AI) techniques have been explored for credit scoring . Although there are no consistent conclusions on which ones are better, recent studies suggest combining multiple classifiers, i.e., ensemble learning, may have a better performance. In our study, using the german credit data available on the UCI Machine Learning Repository we assessed the performance of different Machine learning techniques based on their overall accuracy, but not only, since we supported our results with accuracy measures such as AUC, F1-score, KS and Gini . Our findings suggest that random forest model with tuning parameters , support vector machine based on vanilladot Linear kernel function and lasso regression, with a good overall accuracy values lead us to the most performant AUC values, 0.7833 , 0.7794 and 0.7675, respectively.

Keywords

scoring , credit risk , german data, classifiers , accuracy , AUC.

1 Introduction

The purpose of credit scoring is to classify the applicants into two types : applicants with good credit and applicants with bad credits. When a bank receives a loan application , applicants with good credit have great possibility to repay financial obligation. Applicants with bad credit have high possibility of defaulting. The accuracy of credit scoring is critical to financial institutions profitability . Even 1% improvement on the accuracy of credit scoring of applicants with bad credit will decreases a great loss for financial institutions (Hand & Henley, 1997).

This study aims at adressing this classification problem by using the the applicant's demographic and socio-economic profiles of **german credit data** to examine the risk of lending loan to the customer. We assessed the performance of different Machine learning algorithms (Logistic regression model , Decision tree, random forests, Support vector machines ,neural networks, Lasso) in terms of overall accuracy. For the model optimization, we conducted a comparative assessment of different models combining the effects of Gini, KS, F1-score, balanced accuracy and the area under the ROC curve (AUC) values.

The remainder of the report is organized as follow. In section 2, a general insight of the **Dataset** is presented, with a particular attention to the Data exploration. In section 3, we perform the **Data Pre-processing**. Section 4 explains the **Methods and Analysis** over different Machine learning techniques we used and presents the metrics for the models performance evaluation, while section 5 contains our main **Results**. Section 6 draws **Conclusions and suggestions**.

2 Dataset

2.1 Overview

```
 "german credit data"

Observations: 1,000
Variables: 21
$ Duration                  <dbl> 6, 48, 12, 42, 24, 36, 24, 36, 12, 3...
$ Amount                    <dbl> 1169, 5951, 2096, 7882, 4870, 9055, ...
$ InstallmentRatePercentage <dbl> 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 3, ...
$ ResidenceDuration         <dbl> 4, 2, 3, 4, 4, 4, 4, 2, 4, 2, 1, 4, ...
$ Age                       <dbl> 67, 22, 49, 45, 53, 35, 53, 35, 61, ...
$ NumberExistingCredits     <dbl> 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, ...
$ NumberPeopleMaintenance   <dbl> 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, ...
$ Telephone                 <fct> none, yes, yes, yes, yes, none, yes,...
$ ForeignWorker             <fct> yes, yes, yes, yes, yes, yes, yes, y...
$ CheckingAccountStatus     <fct> lt.0, 0.to.200, none, lt.0, lt.0, no...
$ CreditHistory             <fct> Critical, PaidDuly, Critical, PaidDu...
$ Purpose                   <fct> Radio.Television, Radio.Television, ...
$ SavingsAccountBonds       <fct> Unknown, lt.100, lt.100, lt.100, lt....
$ EmploymentDuration        <fct> gt.7, 1.to.4, 4.to.7, 4.to.7, 1.to.4...
$ Personal                  <fct> Male.Single, Female.NotSingle, Male....
```

```
$ OtherDebtorsGuarantors    <fct> None, None, None, Guarantor, None, N...
$ Property                  <fct> RealEstate, RealEstate, RealEstate, ...
$ OtherInstallmentPlans     <fct> None, None, None, None, None, None, ...
$ Housing                   <fct> Own, Own, Own, ForFree, ForFree, For...
$ Job                       <fct> SkilledEmployee, SkilledEmployee, Un...
$ Class                     <fct> Good, Bad, Good, Good, Bad, Good, Go...
```

The German Credit data is a dataset provided by Dr. Hans Hofmann of the University of Hamburg. It's a publically available from the UCI Machine Learning repository at the following link: https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data). However, this dataset is also built-in in some R packages, like caret , rchallenge or evtree. We decided to use directly the german data from the rchallenge package , which is a transformed version of the GermanCredit dataset with factors instead of dummy variables.

Getting a glimpse of german data, we observe it's a data frame containing 21 variables for a total of 1,000 observations . The response variable (**or outcome,y**) in the dataset corresponds to the **Class** label, which is a binary variable indicating credit risk or credit worthiness with levels Good and Bad. Here, we are going to describe the 20 features and their characteristics:

quantitative features

-*Duration* : duration in months.

-*Amount*: credit Amount.

-*InstallmentRatePercentage*: installment rate in percentage of disposable income.

-*ResidenceDuration*: present residence since .

-*Age* : Client's age.

-*NumberExistingCredits*: number of existing credits at this bank.

-*NumberPeopleMaintenance*: number of people being liable to provide maintenance.

qualitative features

-*Telephone*: binary variable indicating if customer has a registered telephone number

-*ForeignWorker*: binary variable indicating if the customer is a foreign worker

-*CheckingAccountStatus*: factor variable indicating the status of checking account

-*CreditHistory*: factor variable indicating credit history

-*Purpose*: factor variable indicating the credit's purpose

-*SavingsAccountBonds*: factor variable indicating the savings account/bonds

-*EmploymentDuration*: ordered factor indicating the duration of the current employment

-*Personal*: factor variable indicating personal status and sex

-*OtherDebtorsGuarantors*: factor variable indicating Other debtors

-*Property*: factor variable indicating the client's highest valued property

-*OtherInstallmentPlans*: factor variable indicating other installment plans

-*Housing*: factor variable indicating housing

-*Job*: factor indicating employment status

2.2 Data exploration

The Data exploration step disclosed the following findings :

- **Outcome(Class)** : Visual exploration of the outcome, the credit worthiness, shows that there are more people with a good risk than a bad one. In fact, 70% of applicants have a good credit risk while about 30% have a bad credit risk .Then, class are unbalanced and for our subsequent analysis the data splitting in training set and test set should be done with a stratified random sampling method.

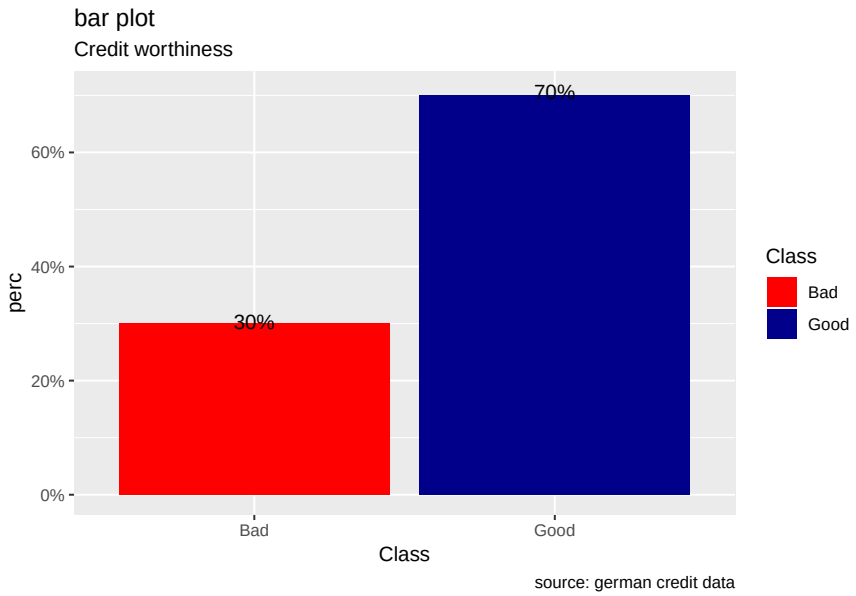


Figure 1: Bar plot of credit worthiness

- **Quantitative features(Age,Duration,Amount,InstallmentRatePercentage,etc)** .The Data exploration of numerical attributes pointed out the subsequent insights :

  ° ***Age*** : From the age variable, we see that the median value for bad risk records is lesser than that of good records, it might be premature to say young people tend to have bad credit records, but we can safely assume it tends to be riskier.

  ° ***Duration, Amount*** : Duration in months as well as credit Amount , appears to show higher median value for bad risk with respect to good risk. Also, the range of these two variables, are more wide for the bad risk records than the good ones. When plotting their density curve along the vertical line for their mean value, we observe that neither Duration in months nor Amount credit is normally distributed. Data tends to show a right skewed distribution especially for the Amount credit variable.

  ° ***InstallmentRatePercentage*** : The bar plot of installment rate shows a significant difference between good and bad credits risk . The number of good records seem to be the double of bad records. When we look at the Box plot, it reveals that the median value for bad records is higher than the good ones even if the two Classes appear to have the same range.

  ° ***NumberExistingCredits, ResidenceDuration, NumberPeopleMaintenance*** : we almost bring up a similar observation for these attributes. While they have their two credit worthiness class "good/bad" which seem to show the same range(cf. boxplot) , good records are always higher than the bad ones(cf bar plot). When we compare some of descriptive statistics of two risks for each of these variable, we observe how mean, median, and sd are almost equal.

Here, in the pdf report we just report the figure for the credit Amount attribute. See all plots and details in the Rmd report at this link : https://mono33.github.io/GermanCreditRisk/Report_GermanRisk.html

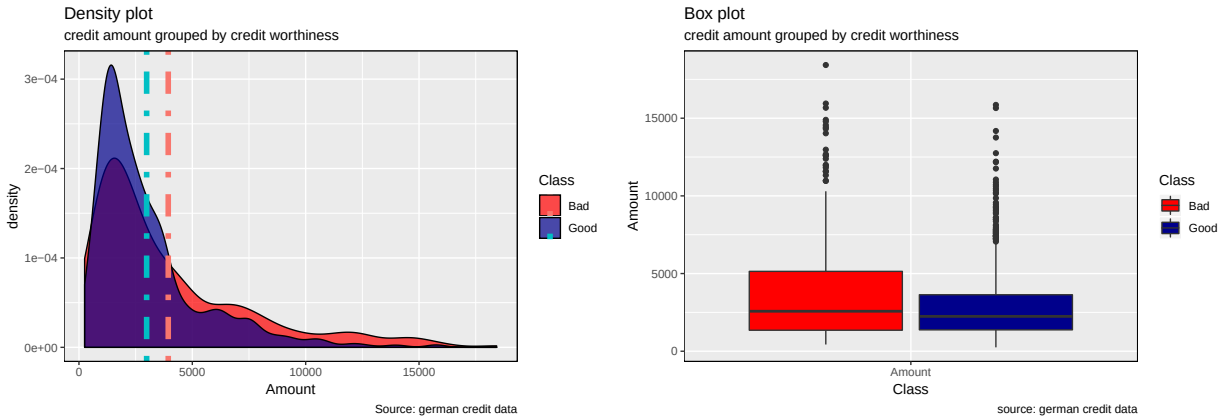


Figure 2: Credit amount accross credit risk, density and box plots

- **Qualitative features(CheckingAccountStatus,CreditHistory,Purpose,SavingAccount,etc)** .The Data exploration of categorical attributes pointed out the subsequent insights :

  ° ***CheckingAccountStatus*** : the current status of the checking account matters as the frequency of the response variables is seen to differ from one sub category to another. Accounts with less than 0 DM houses more number of bad credit risk records while accounts with more than 200 DM the least.we see how applicants with no checking account house the highest number of good records. (NB: DM -> Deutsche Mark)

  ° ***CreditHistory***: Generally, credit history's levels tend to show a higher number of good credit risk than bad credit risk. We observe that proportion of the response variable ,the credit worthiness, changes varies significantly. For categories "NoCredit.AllPaid" and "ThisBank.AllPaid", we see how the number of bad credit records are greater.

  ° ***Purpose***: For the purpose variable, we observe that the proportion of good and bad credit risk varies significantly across the different credit's purposes. While the majority of categories show number of good risk records always greater than the bad ones, the categories as DomesticAppliance ,Education , Repairs and others seem to include more risky records. (the difference between the number of bad and good records is really minimal) (see histogram and ballon plot)

  ° Generally, for the other categorical attributes, we observe a similar trend for which good creditworthiness records are greater than bad ones. Variables like SavingAccountBonds, EmploymentDuration, Personal, ForeignWorker, Housing, Property, or Telephone straightly follow that observation. However, the trend looks more significant for the attributes ***SavingAccountBonds***, telephone,foreign workers, or also EmploymentDuration. Instead, the two features ***OtherInstallmentPlans*** and ***OtherDebtorsGuarantors*** show more risky records. (see on barplot levels Coapplicant or Guarantor for OtherDebtors , and Stores or Bank for OtherInstall)

Here, in the pdf report we just report figures for the Purpose attribute. See all plots and details in the Rmd report at this link.

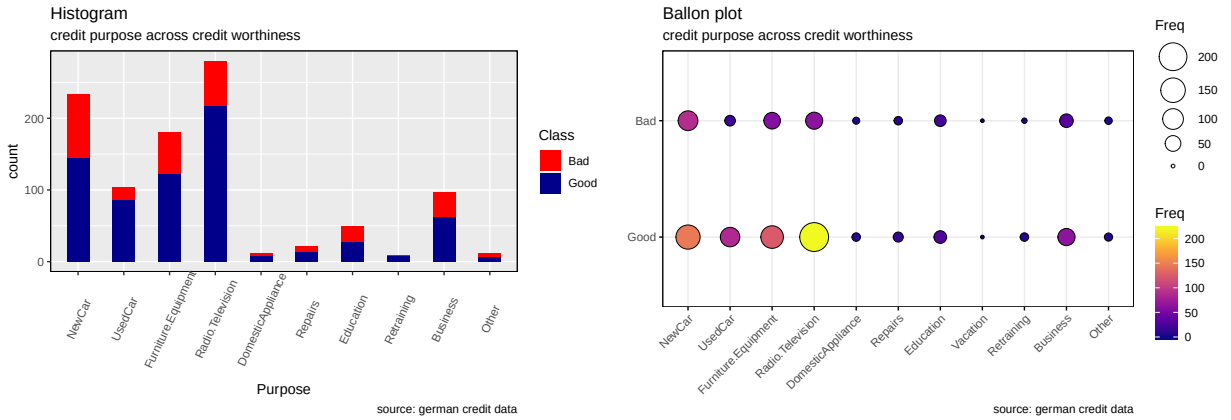


Figure 3: Credit purpose accross credit worthiness, histogram and balloon plots

- **Multivariate Analysis**

Relationship between quantitative and qualitative features, it's performed to understand interactions between different fields in the dataset or finding interactions between variables more than 2 (- ex: pair plot and 3D scatter plot). We performed Multi-panel conditioning focusing on the relationship between Age , Amount credit with respect to Purpose and Personal status/sex (i1 , i2) , we also drew a histogram plot of Amount ~ Age conditioned on Personal status/sex variable (i3).

As in Sivakumar(2015), Multivariate data analysis of the latter revealed the following:

- (i1), ***Age vs credit amount for various purpose*** : As we can see in the first plot ,most of the loans are sought to buy: new car , furniture/equipment, radio/television. It also reveals that surprisingly few people buying used cars have bad rating! And not surprisingly, lower the age of the lonee and higher loan amount correlates to bad credits.
- (i2), ***Age vs credit Amount for Personal status and sex*** : Male: the second plot reveals that single males tend to borrow more, and as before, younger they are and higher the loan amount corresponds to a bad rating. Instead, married/widowed or divorced/separated Males have shown the least amount of borrowing. Because of this, its difficult to visually observe any trends in these categories. Female: the first obvious observation for females is the absence of data for single women. Its not sure if its lack of data or there were no single women applying for loans- though the second possibility seems unlikely in real life. The next most borrowing after single males category is Female not single :divorced/separated/married. The dominant trend in this category is smaller loan amount, higher the age, better the credit rating.
- (i3), ***Distribution Amount ~ Age conditioned on Personal status/sex*** :The histogram reveals that there is a right skewed nearly normal trend seen across all Personal Status and Sex categories, with 30 being the age where people in the sample seem to be borrowing the most.

In the pdf report we just report Multi-panel conditioning for the *Age vs credit Amount for Personal status and sex* . Follow the rmd report to see all plots and details.

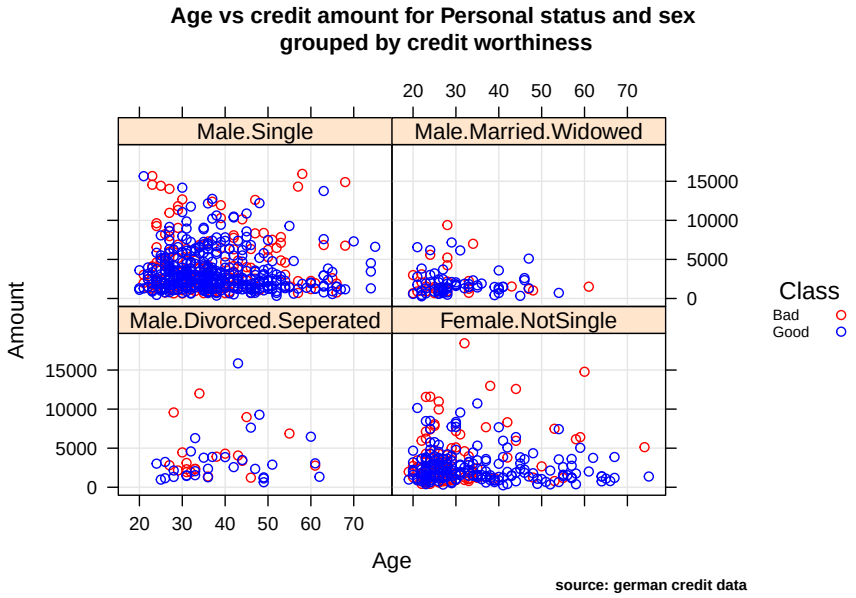


Figure 4: Age vs credit amount for Personal status and sex

3 Data Preprocessing

Real-life data typically needs to be preprocessed (e.g. cleansed, filtered, transformed) in order to be used by the machine learning techniques in the analysis step.

Credit scoring portfolios are frequently voluminous and they are in the range of several thousand, well over 100000 applicants measured on more than 100 variables are quite common (Hand and Hen ley 1997). These portfolios are characterized by noise, missing values, complexity of distributions and by redundant or irrelevant features (Piramuthu 2006). Clearly, the applicants characteristics will vary from situation to situation: an applicant looking for a small loan will be asked for different information from another who is asking for a big loan. Also, according to Garcia et al(2012), in credit scoring applications the resampling approaches have produced important gains in performance when compared to the use of the imbalanced data sets

Then in this section, we'll focus mainly on data preprocessing techniques that are of particular importance when performing a credit scoring task. These techniques include Data wrangling(cleansing, filtering, renaming variables, recodifing factors,etc) , Features selection(filter,wrapper methods) and Data partitioning (stratified random sampling method to correct the imbalanced class).

3.1 Data wrangling

To perform the different steps of data pre-processing part and successive analysis , we firstly make a copy of the german credit dataset, named **german copy**, in order to to keep unchanged our original german credit data .

3.1.1 Missing values

We used the VIM package to explore the data and the structure of the missing or imputed values. Fortunately, We observe that there are no missing values. It's confirmed not only by the *proportion of missings plot* , but also by the *variable sorted by number of missings* output, where each variable have a null count. If they were missing values, it would be presented with red cells like the case of german credit risk from kaggle downloaded at this link (see second plot in rmd report)

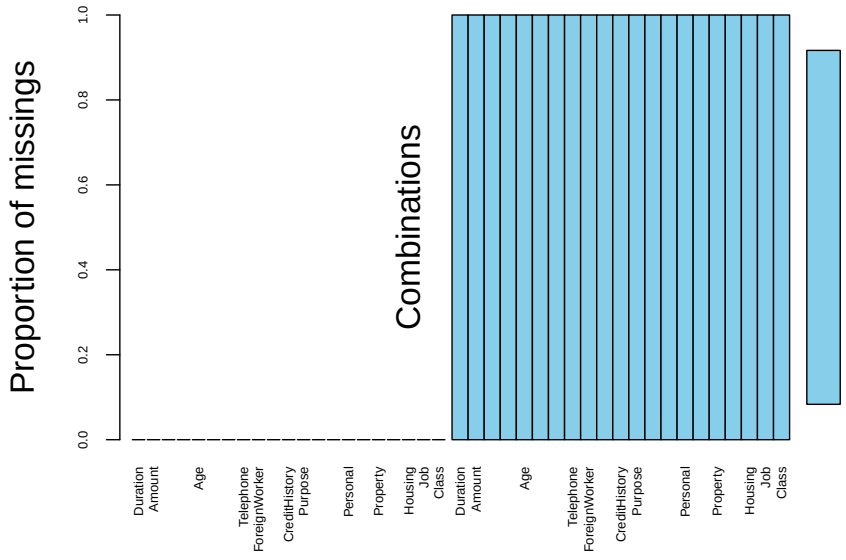


Figure 5: Amount of missing/imputed values in each variable

```
Variables sorted by number of missings:
                 Variable Count
                 Duration     0
                   Amount     0
InstallmentRatePercentage     0
        ResidenceDuration     0
                      Age     0
    NumberExistingCredits     0
  NumberPeopleMaintenance     0
                Telephone     0
            ForeignWorker     0
    CheckingAccountStatus     0
            CreditHistory     0
                  Purpose     0
      SavingsAccountBonds     0
       EmploymentDuration     0
                 Personal     0
   OtherDebtorsGuarantors     0
                 Property     0
    OtherInstallmentPlans     0
                 Housing     0
                     Job     0
                   Class     0
```

3.1.2 Renaming variables

Renaming data columns is a common task in Data pre-processing that can make implementing ML algorithms/ writing code faster by using short, intuitive names. We used the dplyr function rename() to make it easy. See the new variable names (eg. Class -> Risk, CreditHistory -> credit_hist)

```
 "german copy"
```

```
Observations: 1,000
Variables: 21
$ Duration         <dbl> 6, 48, 12, 42, 24, 36, 24, 36, 12, 30, 12, 48...
$ Amount           <dbl> 1169, 5951, 2096, 7882, 4870, 9055, 2835, 694...
$ installment_rate <dbl> 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 3, 1, 4, 2, ...
$ present_resid    <dbl> 4, 2, 3, 4, 4, 4, 4, 2, 4, 2, 1, 4, 1, 4, 4, ...
$ Age              <dbl> 67, 22, 49, 45, 53, 35, 53, 35, 61, 28, 25, 2...
$ n_credits        <dbl> 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, ...
$ n_people         <dbl> 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ Telephone        <fct> none, yes, yes, yes, yes, none, yes, none, ye...
$ ForeignWorker    <fct> yes, yes, yes, yes, yes, yes, yes, yes, yes, ...
$ check_acct       <fct> lt.0, 0.to.200, none, lt.0, lt.0, none, none,...
$ credit_hist      <fct> Critical, PaidDuly, Critical, PaidDuly, Delay...
$ Purpose          <fct> Radio.Television, Radio.Television, Education...
$ savings_acct     <fct> Unknown, lt.100, lt.100, lt.100, lt.100, Unkn...
$ present_emp      <fct> gt.7, 1.to.4, 4.to.7, 4.to.7, 1.to.4, 1.to.4,...
$ status_sex       <fct> Male.Single, Female.NotSingle, Male.Single, M...
$ other_debt       <fct> None, None, None, Guarantor, None, None, None...
$ Property         <fct> RealEstate, RealEstate, RealEstate, Insurance...
$ other_install    <fct> None, None, None, None, None, None, None, Non...
$ Housing          <fct> Own, Own, Own, ForFree, ForFree, ForFree, Own...
$ Job              <fct> SkilledEmployee, SkilledEmployee, UnskilledRe...
$ Risk             <fct> Good, Bad, Good, Good, Bad, Good, Good, Good,...
```

3.1.3 Skewness

Skewness is defined to be the third standardized central moment .You can easily tell if a distribution is skewed by simple visualization. There are different ways may help to remove skewness such as log, square root or inverse. However it is often difficult to determine from plots which transformation is most appropriate for correcting skewness. The Box-Cox procedure automatically identified a transformation from the family of power transformations that are indexed by a parameter λ.

This family includes:

- log transformation ($\lambda = 0$)
- square transformation ($\lambda = 2$)
- square root ($\lambda = 0.5$)
- inverse ($\lambda = -1$)

We used preProcess() function in caret to apply this transformation by changing the method argument to **BoxCox**.

However, for the purpose of our study we concluded that it was not useful to normalize all skewed attributes. See all details in the rmd report

Before to continue with the Feature selection step, we recoded the response variable **Risk**, by creating a new variable (numerical) **risk_bin** where **0** corresponds to a Bad credit worthiness and **1** to a Good credit worthiness. We didn't eliminate the original Risk outcome , they can be both useful for successive analysis depending of the Machine learning model we'll test.

```
 "german copy "
```

```
Observations: 1,000
Variables: 22
$ Duration         <dbl> 6, 48, 12, 42, 24, 36, 24, 36, 12, 30, 12, 48...
$ Amount           <dbl> 1169, 5951, 2096, 7882, 4870, 9055, 2835, 694...
$ installment_rate <dbl> 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 3, 1, 4, 2, ...
$ present_resid    <dbl> 4, 2, 3, 4, 4, 4, 4, 2, 4, 2, 1, 4, 1, 4, 4, ...
$ Age              <dbl> 67, 22, 49, 45, 53, 35, 53, 35, 61, 28, 25, 2...
$ n_credits        <dbl> 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, ...
$ n_people         <dbl> 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ Telephone        <fct> none, yes, yes, yes, yes, none, yes, none, ye...
$ ForeignWorker    <fct> yes, yes, yes, yes, yes, yes, yes, yes, yes, ...
$ check_acct       <fct> lt.0, 0.to.200, none, lt.0, lt.0, none, none,...
$ credit_hist      <fct> Critical, PaidDuly, Critical, PaidDuly, Delay...
$ Purpose          <fct> Radio.Television, Radio.Television, Education...
$ savings_acct     <fct> Unknown, lt.100, lt.100, lt.100, lt.100, Unkn...
$ present_emp      <fct> gt.7, 1.to.4, 4.to.7, 4.to.7, 1.to.4, 1.to.4,...
$ status_sex       <fct> Male.Single, Female.NotSingle, Male.Single, M...
$ other_debt       <fct> None, None, None, Guarantor, None, None, None...
$ Property         <fct> RealEstate, RealEstate, RealEstate, Insurance...
$ other_install    <fct> None, None, None, None, None, None, None, Non...
$ Housing          <fct> Own, Own, Own, ForFree, ForFree, ForFree, Own...
$ Job              <fct> SkilledEmployee, SkilledEmployee, UnskilledRe...
$ Risk             <fct> Good, Bad, Good, Good, Bad, Good, Good, Good,...
$ risk_bin         <dbl> 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, ...
```

3.2 Features selection

According to HA & Nguyen (2016), there are two different categories of feature selection methods: filter and wrapper methods.

3.2.1 filter methods

The filter approach considers the feature selection process as a precursor stage of learning algorithms. The filter model uses evaluation functions to evaluate the classification performances of subsets of features. There are many evaluation functions such as feature importance, Gini, information gain, filtering, etc. A disadvantage of this approach is that there is no direct relationship between the feature selection process and the performance of learning algorithms.

- **sbf (selection by filtering, caret package):**

The output of Model fitting after applying sbf univariate filters tell us that the top 5 selected variables are : Age (100%), Amount (100%), check_acct0.to.200 (100%), check_acctnone (100%), credit_histCritical (100%).Let's try another method to assert or rebut the latter findings. For more details about selection by filtering, read this.

```
Selection By Filter

Outer resampling method: Cross-Validated (10 fold, repeated 5 times)

Resampling performance:

 Accuracy  Kappa AccuracySD KappaSD
    0.743 0.3189    0.03666 0.09716

Using the training set, 23 variables were selected:
   Duration, Amount, installment_rate, Age, ForeignWorkeryes...

During resampling, the top 5 selected variables (out of a possible 30):
   Age (100%), Amount (100%), check_acct0.to.200 (100%), check_acctnone (100%), credit_histCritical (10

On average, 21.8 variables were selected (min = 19, max = 25)
```

- **WOE,IV (Information package) :**

Weight of Evidence(WOE): WOE shows predictive power of an independent variable in relation to dependent variable. It evolved with credit scoring to magnify separation power between a good customer and a bad customer, hence it is one of the measures of separation between two classes(good/bad, yes/no, 0/1, A/B, response/no-response). It is defined as:

$$WOE = ln(\frac{Distribution.of.nonEvents(Good)}{Distribution.of.Events(Bad)})$$

.It is computed from the basic odds ratio: (Distribution of Good Credit Outcomes) / (Distribution of Bad Credit Outcomes)

Some benefits of WOE lie in the fact that it can treat outliers (Suppose you have a continuous variable such as annual salary and extreme values are more than 500 million dollars. These values would be grouped to a class of (let's say 250-500 million dollars). Later, instead of using the raw values, we would be using WOE scores of each classes.) and handle missing values as missing values can be binned separately. Also, It handles both continuous and categorical variables so there is no need for dummy variables.

Information value(IV): Information value is one of the most useful technique to select important variables in a predictive model. It helps to rank variables on the basis of their importance. The IV is calculated using the following formula :

$$IV = \sum(\%nonEvents - \%events) * WOE$$

According to Siddiqi (2006), by convention the values of the IV statistic in credit scoring can be interpreted as follows.

If the IV statistic is:

- Less than 0.02, then the predictor is not useful for modeling (separating the Goods from the Bads)
- 0.02 to 0.1, then the predictor has only a weak relationship to the Goods/Bads odds ratio
- 0.1 to 0.3, then the predictor has a medium strength relationship to the Goods/Bads odds ratio
- 0.3 to 0.5, then the predictor has a strong relationship to the Goods/Bads odds ratio.
- greater than 0.5, suspicious relationship (Check once)

```
"Get tables from the IV statistic - see long output in the rmd report"
```

```
 "Get summary of the IV statistic"
```

```
           Variable    IV
10       check_acct 66.60
11      credit_hist 29.32
1          Duration 27.79
13     savings_acct 19.60
12          Purpose 16.92
5               Age 12.12
17         Property 11.26
2            Amount 11.18
14      present_emp  8.64
19          Housing  8.33
18    other_install  5.76
15       status_sex  4.47
9     ForeignWorker  4.39
16       other_debt  3.20
3  installment_rate  2.63
6         n_credits  1.01
20              Job  0.88
8         Telephone  0.64
4     present_resid  0.36
7           n_people  0.00
```

```
 "very very weak predictors (IV< 2%)- shall exclude them from modelling"
```

Variable	IV
n_credits	**1.01**
Job	**0.88**
Telephone	**0.64**
present_resid	**0.36**
n_people	**0.00**

```
 "very weak predictors (2%<=IV< 10%)-may or may not include them while modeling"
```

Variable	IV
present_emp	**8.64**
Housing	**8.33**
other_install	**5.76**
status_sex	**4.47**
ForeignWorker	**4.39**
other_debt	**3.20**
installment_rate	**2.63**

```
 "medium prediction power (10%<=IV< 30%)"
```

Variable	IV
credit_hist	**29.32**
Duration	**27.79**
savings_acct	**19.60**
Purpose	**16.92**
Age	**12.12**
Property	**11.26**
Amount	**11.18**

```
"no strong predictor with IV between 30% to 50%"
```

Variable	IV

```
"very high prediction power (IV > 50%)-suspicious and require further investigation"
```

Variable	IV
check_acct	66.6

3.2.2 wrapper methods

Wrapper methods consider the selection of a set of features as a search problem, where different combinations are prepared, evaluated and compared to other combinations. A predictive model us used to evaluate a combination of features and assign a score based on model accuracy. The search process may be methodical such as a best-first search, it may stochastic such as a random hill-climbing algorithm, or it may use heuristics, like forward and backward passes to add and remove features. One disadvantage of the wrapper approach is highly computational cost.

An example of a wrapper method is the recursive feature elimination algorithm. Another example is the boruta algorithm.

- **rfe (recursive feature elimination, caret package):**

Recursive feature elimination via the caret package, is a wrapper method which is explained by the algorithm below. For further details follow Recursive Feature Elimination via caret

Algorithm 2: Recursive feature elimination incorporating resampling

```
for Each Resampling Iteration do
    Partition data into training and test/hold–back set via resampling
    Tune/train the model on the training set using all predictors
    Predict the held–back samples
    Calculate variable importance or rankings
    for Each subset size S_i, i = 1 ... S do
        Keep the S_i most important variables
        [Optional] Pre–process the data
        Tune/train the model on the training set using S_i predictors
        Predict the held–back samples
        [Optional] Recalculate the rankings for each predictor
    end
end
Calculate the performance profile over the S_i using the held–back samples
Determine the appropriate number of predictors
Estimate the final list of predictors to keep in the final model
Fit the final model based on the optimal S_i using the original training set
```

Figure 6: recursive feature elimitation algorithm

The recursive feature selection output (Outer resampling method: Cross-Validated (20 fold)) shows that the top 5 variables are: check_acct, Duration, credit_hist, Amount and savings_acct. These findings are somehow in line with those of WOE and IV statistics. (check_acct has the highest power prediction, and features such as Amount,Duration, savings_acct have medium prediction power) For more details on how we build our recursive feature selection algorithm, follow the rmd report at this link.

```
 "Cluster stopped."


Recursive feature selection

Outer resampling method: Cross-Validated (20 fold)

Resampling performance over subset size:

 Variables Accuracy   Kappa AccuracySD KappaSD Selected
         1    0.687 0.01115    0.03197 0.03603
         2    0.727 0.26590    0.04169 0.13144
         3    0.737 0.28146    0.04269 0.11063
         4    0.729 0.30158    0.05524 0.15665
         5    0.746 0.34942    0.05236 0.14191
         6    0.772 0.40465    0.05085 0.14150
         7    0.770 0.39423    0.04026 0.12075
         8    0.766 0.38039    0.05586 0.16309
         9    0.770 0.39346    0.05712 0.16442
        10    0.776 0.40547    0.05295 0.15160
        11    0.778 0.40704    0.05386 0.15556        *
        12    0.773 0.39260    0.04953 0.14876
        13    0.772 0.38039    0.06031 0.17823
        14    0.772 0.38123    0.05288 0.15897
        15    0.760 0.34651    0.05351 0.15822
        16    0.772 0.37718    0.05167 0.15584
        17    0.769 0.37059    0.05004 0.15479
        18    0.761 0.35214    0.04564 0.14335
        19    0.773 0.37491    0.04366 0.13925
        20    0.771 0.37221    0.04745 0.14784

The top 5 variables (out of 11):
   check_acct, Duration, credit_hist, Amount, savings_acct
```

- **Boruta algorithm:**

Boruta algorithm is a wrapper built around the random forest classification algorithm implemented in the R package randomForest (Liaw and Wiener, 2002). It tries to capture all the important, interesting features you might have in your dataset with respect to an outcome variable. To Read all steps of this algorithm follow Kursa & Rudnicki (2010), or rmd report.

Boruta function uses y , the response vector , as factor . Then in the formula, we considered Risk and not risk_bin. We observed that Boruta performed 99 iterations in less than 2 minutes, and the confirmed important attributes are 11 : Duration , Amount , Age , check_acct , credit_hist , Purpose , savings_acct , present_emp , other_debt , Property and other_install (results that go in accordance with WOE and IV statistics, from weak to highest predictors) . See the plot below, where Blue boxplots correspond to minimal, average and maximum Z score of a shadow attribute. Red and green boxplots represent Z scores of respectively rejected and confirmed attributes. Yellow boxplots represent Z scores of 5 tentative attributes left.

```
Boruta performed 99 iterations in 1.025517 mins.
 11 attributes confirmed important: Age, Amount, check_acct,
credit_hist, Duration and 6 more;
```

```
 4 attributes confirmed unimportant: ForeignWorker, n_people,
status_sex, Telephone;
 5 tentative attributes left: Housing, installment_rate, Job,
n_credits, present_resid;

Risk ~ Duration + Amount + Age + check_acct + credit_hist + Purpose +
    savings_acct + present_emp + other_debt + Property + other_install
<environment: 0x00000000311eaad8>
```

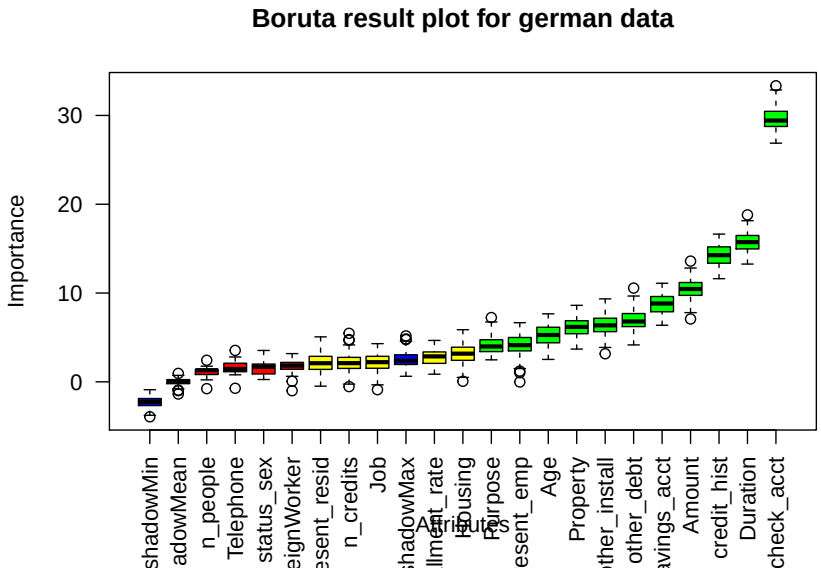


Figure 7: Boruta result plot

3.2.3 Embedded methods

A third feature selection method, *embedded methods* , is detailed in Aggarwal (2014). This class of methods embedding feature selection with classifier construction, have the advantages of wrapper models — they include the interaction with the classification model and filter models—and they are far less computationally intensive than wrapper methods. There are of 3 types : The first are pruning techniques that first utilize all features to train a model and then attempt to eliminate some features by setting the corresponding coefficients to 0 , while maintaining model performance such as recursive feature elimination using a support vector machine . The second are models with a built-in mechanism for feature selection such as C4.5 algorithm. The third are regularization models with objective functions that minimize fitting errors and in the meantime force the coefficients to be small or to be exactly zero (Elastic Net or Ridge Regression)

However in our study, we are not going to developp these models as features selectors . Here, we just give an overview of this third feature selection approach.

At the end, Based on the Weight of Evidence and Information Value statistics results, we kept attributes which $IV > 2$. More over, the wrapper methods , recursive feature elimination and Boruta algorithm confirm this choice since they helped to investigate better on the suspicious character of the highest power predictor, check_acct. The important attributes validated by these algorithms corroborate with our choice.

With the latter attributes ($IV > 2$) and the target *Risk*, we created a new copy of our dataset, **german.copy2** . For convenience, we made a further recoding of the outcome, Risk. we changed its levels, Good = 1 and Bad = 0.

```
 "german.copy2"
```

```
Observations: 1,000
Variables: 16
$ check_acct       <fct> lt.0, 0.to.200, none, lt.0, lt.0, none, none,...
$ credit_hist      <fct> Critical, PaidDuly, Critical, PaidDuly, Delay...
$ Duration         <dbl> 6, 48, 12, 42, 24, 36, 24, 36, 12, 30, 12, 48...
$ savings_acct     <fct> Unknown, lt.100, lt.100, lt.100, lt.100, Unkn...
$ Purpose          <fct> Radio.Television, Radio.Television, Education...
$ Age              <dbl> 67, 22, 49, 45, 53, 35, 53, 35, 61, 28, 25, 2...
$ Property         <fct> RealEstate, RealEstate, RealEstate, Insurance...
$ Amount           <dbl> 1169, 5951, 2096, 7882, 4870, 9055, 2835, 694...
$ present_emp      <fct> gt.7, 1.to.4, 4.to.7, 4.to.7, 1.to.4, 1.to.4,...
$ Housing          <fct> Own, Own, Own, ForFree, ForFree, ForFree, Own...
$ other_install    <fct> None, None, None, None, None, None, None, Non...
$ status_sex       <fct> Male.Single, Female.NotSingle, Male.Single, M...
$ ForeignWorker    <fct> yes, yes, yes, yes, yes, yes, yes, yes, yes, ...
$ other_debt       <fct> None, None, None, Guarantor, None, None, None...
$ installment_rate <dbl> 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 3, 1, 4, 2, ...
$ Risk             <fct> 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, ...
```

3.3 Data partitioning

Stratified random sampling is a method of sampling that involves the division of a population into smaller groups known as strata. This is useful for imbalanced datasets, and can be used to give more weight to a minority class. In stratified random sampling, the strata are formed based on members' shared attributes or characteristics. See more here or read Garcia et al(2012) that is already mentioned at the introduction of Data pre-processing paragraph.

In our case we used Good/Bad as strata and partition data into 70%-30% as train and test sets. The caret function **createDataPartition()** can be used to create balanced splits of the data

```
 "training set"
```

```
Observations: 700
Variables: 16
$ check_acct       <fct> lt.0, 0.to.200, none, none, 0.to.200, 0.to.20...
$ credit_hist      <fct> Critical, PaidDuly, Critical, PaidDuly, PaidD...
$ Duration         <dbl> 6, 48, 12, 36, 36, 12, 48, 24, 30, 24, 9, 6, ...
$ savings_acct     <fct> Unknown, lt.100, lt.100, Unknown, lt.100, lt....
$ Purpose          <fct> Radio.Television, Radio.Television, Education...
$ Age              <dbl> 67, 22, 49, 35, 35, 25, 24, 53, 25, 31, 48, 4...
$ Property         <fct> RealEstate, RealEstate, RealEstate, Unknown, ...
$ Amount           <dbl> 1169, 5951, 2096, 9055, 6948, 1295, 4308, 242...
$ present_emp      <fct> gt.7, 1.to.4, 4.to.7, 1.to.4, 1.to.4, lt.1, l...
$ Housing          <fct> Own, Own, Own, ForFree, Rent, Rent, Rent, Own...
$ other_install    <fct> None, None, None, None, None, None, None, Non...
$ status_sex       <fct> Male.Single, Female.NotSingle, Male.Single, M...
$ ForeignWorker    <fct> yes, yes, yes, yes, yes, yes, yes, yes, yes, ...
$ other_debt       <fct> None, None, None, None, None, None, None, Non...
$ installment_rate <dbl> 4, 2, 2, 2, 2, 3, 3, 4, 2, 3, 4, 2, 1, 3, 2, ...
$ Risk             <fct> 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, ...
```

	Count	Percentage
0	210	30
1	490	70

```
 "test set"

Observations: 300
Variables: 16
$ check_acct       <fct> lt.0, lt.0, none, none, 0.to.200, 0.to.200, l...
$ credit_hist      <fct> PaidDuly, Delay, PaidDuly, PaidDuly, Critical...
$ Duration         <dbl> 42, 24, 24, 12, 30, 12, 24, 15, 24, 24, 6, 6,...
$ savings_acct     <fct> lt.100, lt.100, 500.to.1000, gt.1000, lt.100,...
$ Purpose          <fct> Furniture.Equipment, NewCar, Furniture.Equipm...
$ Age              <dbl> 45, 53, 53, 61, 28, 22, 60, 28, 32, 44, 36, 3...
$ Property         <fct> Insurance, Unknown, Insurance, RealEstate, Ca...
$ Amount           <dbl> 7882, 4870, 2835, 3059, 5234, 1567, 1199, 140...
$ present_emp      <fct> 4.to.7, 1.to.4, gt.7, 4.to.7, Unemployed, 1.t...
$ Housing          <fct> ForFree, ForFree, Own, Own, Own, Own, Own, Re...
$ other_install    <fct> None, None, None, None, None, None, None, Non...
$ status_sex       <fct> Male.Single, Male.Single, Male.Single, Male.D...
$ ForeignWorker    <fct> yes, yes, yes, yes, yes, yes, yes, yes, yes, ...
$ other_debt       <fct> Guarantor, None, None, None, None, None, None...
$ installment_rate <dbl> 2, 3, 3, 2, 4, 1, 4, 2, 4, 4, 1, 4, 3, 3, 2, ...
$ Risk             <fct> 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, ...
```

	Count	Percentage
0	90	30
1	210	70

The data preprocessing phase is usually not definitive because it requires a lot of attention and subsequent various explorations on the variables. It must be aimed at obtaining better predictive results and in this sense, the further phases of model evaluations can help us to understand which particular preprocessing approaches are actually indispensable or useful for a specific model purpose.

4 Methods and Analysis

In machine learning, classification is considered an instance of the supervised learning methods,i.e., inferring a function from labeled training data. The training data consist of a set of training examples, where each example is a pair consisting of an input object (typically a vector of features) $x = (x_1, x_2, ..., x_d)$ and a desired output value (typically a class label) $y\{C_1, C_2, ..., C_K\}$. Given such a set of training data, the task of a classification algorithm is to analyze the training data and produce an inferred function, which can be used to classify new (so far unseen) examples by assigning a correct class label to each of them. An example would be assigning an applicant credit scoring into "good" or "bad" classes(then consider the outcome Y as a boolean variable, as in our study). A general process of data classification usually consists of two phases—the training phase and the prediction phase.

There are many classification methods in the literature. These methods can be categorized broadly into probabilistic classification algorithms or linear classifiers (Naive bayes classifiers, logistic regression, Hidden markov models, etc), support vector machines, decision trees, and Neural networks. (Aggarwal, 2014).

In this section, we are going to explain the methodology over differents Machine Learning algorithms we used taking our principal references from Aggarwal(2014), and to present the metrics for the model performance evaluation.

4.1 Evaluated Algorithms

4.1.1 Logistic regression

Logistic regression is an approach for predicting the outcome of a categorialdependent variable based on one or more observed features. The probabilities describing the possible outcomes are modeled as a function of the observed variables using a logistic function.

The goal of logistic regression is to directly estimate the distribution P(Y|X) from the training data. Formally, the logistic regression model is defined as

$$p(Y = 1|X) = g({}^{T}X) = \frac{1}{1 + e^{T}X}, \tag{1}$$

where $g(z) = \frac{1}{1+ez}$ is called the logistic function or the sigmoid function, and ${}^{T}X = {}_{0} + \sum_{i=1}^{d} iXi$. As the sum of the probabilities must equal 1, p(Y = 0|X) can be estimated using the following equation

$$p(Y = 0|X) = 1g({}^{T}X) = \frac{e^{T}X}{1 + e^{T}X}. \tag{2}$$

Because logistic regression predicts probabilities, rather than just classes, we can fit it using likelihood. For each training data point, we have a vector of features, $X = (X_0, X_1, ..., X_d)(X_0 = 1)$, and an observed class, $Y = y_k$. The probability of that class was either $g({}^{T}X)$ if $yk = 1$, or $1 - g({}^{T}X)$ if $y_k = 0$. Note that we can combine both Equation (1) and Equation (2) as a more compact form

$$\begin{aligned} p(Y = y_k|X;) &= (p(Y = 1|X))^{y_k}(p(Y = 0|X))^{1-y_k} \\ &= g({}^{T}X)^{y_k}(1 - g(TX))^{1y_k} \end{aligned}$$

Assuming that the N training examples were generated independently, the likelihood of the parameters can be written as

$$\begin{aligned} L() &= p(\overline{Y}|X;) \\ &= \prod_{n=1}^{N} p(Y^{(n)} = y_k|X^{(n)};)) \\ &= \prod_{n=1}^{N} \left(p(Y^{(n)} = 1|X^{(n)})\right)^{Y^{(n)}} \left(p(Y^{(n)} = 0|X^{(n)})\right)^{1-Y^{(n)}} \\ &= \prod_{n=1}^{N} \left(g({}^{T}X^{(n)})\right)^{Y^{(n)}} \left(1 - g({}^{T}X^{(n)})\right)^{1-Y^{(n)}} \end{aligned} \tag{3}$$

where is the vector of parameters to be estimated, $Y(n)$ denotes the observed value of Y in the nth training example, and $X(n)$ denotes the observed value of X in the nth training example. To classify any given X, we generally want to assign the value y_k to Y that maximizes the likelihood. Maximizing the likelihood is equivalent to maximizing the log likelihood.

In our study, using the **glm()** function of the stats package, we fit our logistic regression model on the training set. Then, in the testing phase, we evaluated our fitted model on the test set with the **predict()** function of the same package.

4.1.2 Decision trees

Decision trees create a hierarchical partitioning of the data, which relates the different partitions at the leaf level to the different classes. The hierarchical partitioning at each level is created with the use of a split criterion. The split criterion may either use a condition (or predicate) on a single attribute, or it may contain a condition on multiple attributes. The former is referred to as a univariate split, whereas the latter is referred to as a multivariate split. The overall approach is to try to recursively split the training data so as to maximize the discrimination among the different classes over different nodes. The discrimination among the different classes is maximized, when the level of skew among the different classes in a given node is maximized. A measure such as the gini-index or entropy is used in order to quantify this skew. For example, if $p_1 ... p_k$ is the fraction of the records belonging to the k different classes in a node N, then the gini-index $G(N)$ of the node N is defined as follows:

$$G(N) = 1 \ - \sum_{i=1}^{k} = p_i^2 \tag{4}$$

The value of $G(N)$ lies between 0 and $11/k$. The smaller the value of G(N), the greater the skew. In the cases where the classes are evenly balanced, the value is $11/k$. An alternative measure is the entropy $E(N)$:

$$E(N) = -\sum_{i=1}^{k} pi * log(pi) \tag{5}$$

The value of the entropy lies between 0 and $log(k)$. The value is $log(k)$, when the records are perfectly balanced among the different classes. This corresponds to the scenario with maximum entropy. The smaller the entropy, the greater the skew in the data. Thus, the gini-index and entropy provide an effective way to evaluate the quality of a node in terms of its level of discrimination between the different classes.

In our study, we built our classification tree model on the training set using the rpart function of the rpart package. By default, **rpart()** function uses the Gini impurity measure to split the node. The higher the Gini coefficient, the more different instances within the node. We predicted the fitted tree on the test set with the predict() function. Major explanations of Recursive Partitioning Using the rpart Routines are given in Therneau & Atkinson (2018).

4.1.3 Random forests

Random Forest can be regarded as a variant of Bagging approach. It follows the major steps of Bagging and uses decision tree algorithm to build base classifiers. Besides Bootstrap sampling and majority voting used in Bagging, Random Forest further incorporates random feature space selection into training set construction to promote base classifiers' diversity.

Random Forest algorithm builds multiple decision trees following the summarized algorithm below and takes majority voting of the prediction results of these trees. To incorporate diversity into the trees, Random Forest approach differs from the traditional decision tree algorithm in the following aspects when building each tree: First, it infers the tree from a Bootstrap sample of the original training set. Second, when selecting the best feature at each node, Random Forest only considers a subset of the feature space. These two modifications of decision tree introduce randomness into the tree learning process, and thus increase the diversity of base classifiers. In practice, the dimensionality of the selected feature subspace n' controls the randomness. If we set $n' = n$ where n is the original dimensionality, then the constructed decision tree is the same as the traditional deterministic one. If we set $n' = 1$, at each node, only one feature will be randomly selected to split the data, which leads to a completely random decision tree.

Specifically, the following describes the general procedure of Random Forest algorithm:

Algorithm 19.4 Random Forest

Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^m$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)
Output: An ensemble classifier H

1: **for** $t \leftarrow 1$ to T **do**
2: Construct a sample data set $\mathcal{D}_t$ by randomly sampling with replacement in $\mathcal{D}$
3: Learn a decision tree h_t by applying
$LearnDecisionTree(\mathcal{D}_t, iteration = 0, ParentNode = root)$:
4: If stop criterion is satisfied, return
5: Randomly sample features in the whole feature space $\mathbb{R}^n$ to get a new data set
$\hat{\mathcal{D}}_{current} = RandomSubset(\mathcal{D}_{current})$
6: Find the best feature q^* according to impurity gain
7: Split data $(\mathcal{D}_L, \mathcal{D}_R) = split(\mathcal{D}_{current}, q^*)$
8: Label the new parent node $v = parent.newchild(q^*)$
9: Conduct $LearnDecisionTree(\mathcal{D}_L, iteration = iteration + 1, ParentNode = v)$ and
$LearnDecisionTree(\mathcal{D}_R, iteration = iteration + 1, ParentNode = v)$
10: **end for**
11: **return** $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbf{1}(h_t(\mathbf{x}) = y)$

Figure 8: random forest algorithm

In our study, we used the **randomForest()** function of the randomForest package which implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code), to build our random forest models on training set . Then , we predicted the latter model on test set with the **predict()** function.

4.1.4 Support Vector Machines

SVM methods use linear conditions in order to separate out the classes from one another. The idea is to use a linear condition that separates the two classes from each other as well as possible. SVMs were developed by Cortes & Vapnik (1995) for binary classification. Their approach may be roughly sketched with the following tasks : Class separation, Overlapping classes, Nonlinearity and Problem solution which is a quadratic optimization problem.

The principal task, Class separation, lies in looking for the optimal separating hyperplane between the two classes by maximizing the margin between the classes' closest points (see Figure below)—the points lying on the boundaries are called support vectors, and the middle of the margin is our optimal separating hyperplane. For further details, follow Meyer and Wien(2015) or Karatzoglou et al (2005).

The package e1071 offers an interface to the award-winning C++- implementation by Chang & Lin(2001), libsvm

In our study, we used the **svm()** function of that package, which was designed to be as intuitive as possible (we also compared with results of **ksvm()** function of the kernlab package). Models are fitted on the training set and predicted on unseen data(test set) as usual.

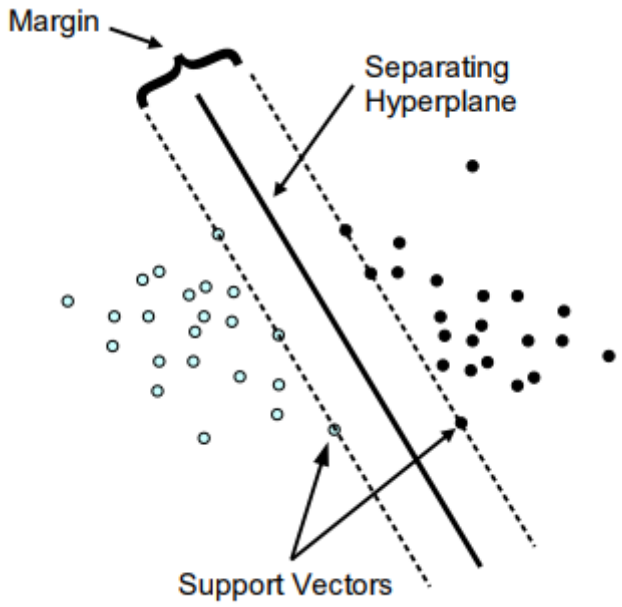


Figure 9: SVM Classification (linear separable case)

4.1.5 Neural Networks

An artificial neural network (ANN) or neural net is a graph of connected units representing a mathematical model of biological neurons. Those units are sometimes referred to as processing units, nodes, or simply neurons. The units are connected through unidirectional or bidirectional arcs with weights representing the strength of the connections between units. This is inspired from the biological model in which the connection weights represent the strength of the synapses between the neurons, inhibiting or facilitating the passage of signals.

The artificial neuron, as illustrated in Figure x, is a computational engine that transforms a set of inputs $x = (x_1, x_2, ..., x_d)$ into a single output o using a composition of two functions as follows:

- A **net value function** ξ, which utilizes the unit's parameters or weights w to summarize input data into a net value, υ, as $\upsilon = \xi(x, w)$. The net value function mimics the behavior of a biological neuron as it aggregates signals from linked neurons into an internal representation. Typically, it takes the form of a *weighted sum*, a *distance*, or a *kernel*.
- An **activation function**, or squashing function, φ, that transforms net value into the unit's output value o as $o = \varphi(\upsilon)$. The activation function simulates the behaviour of a biological neuron as it decides to fire or inhibit signals, depending on its internal logic. The output value is then dispatched to all receiving units as determined by the underlying topology. Various activations have been proposed.The most widely-used ones include the linear function, the step or threshold function, and the sigmoid and hyperbolic tangent function.

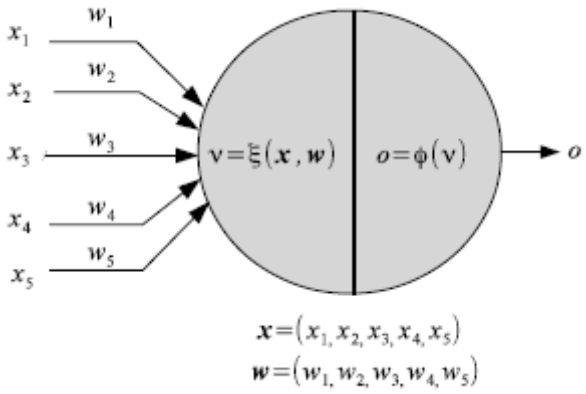


Figure 10: Mathematical model of an ANN unit

Generally we distinguish two types of neural networks : single and multilayer neural networks. The following figures are taken from Aggarwal(2014)

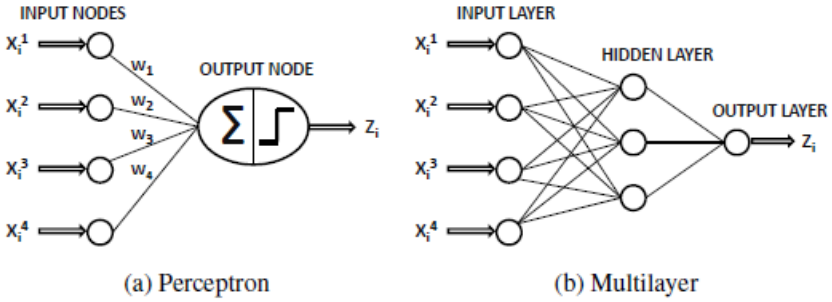


Figure 11: Single and multilayer neural networks.

In our study, we principally used the nnet package by Ripley et al(2016) . We also took into account the NeuralNetTools(Visualization and Analysis Tools for Neural Networks) and neuralnet packages. For a more in-depth view on the exact workings of the neuralnet package, see neuralnet: Training of Neural Networks by F. Günther and S. Fritsch. For the methods in NeuralNetTools, see here. As in Soni & Abdullahi(2015), we built our neural network model on training set; however using the **nnet() function** instead of **neuralnet() function** . We predicted the fitted model on test set with the **predict()** function.

4.1.6 Lasso regression

Lasso, or Least Absolute Shrinkage and Selection Operator, is quite similar conceptually to ridge regression. It also adds a penalty for non-zero coefficients, but unlike ridge regression which penalizes sum of squared coefficients (the so-called L2 penalty), lasso penalizes the sum of their absolute values (L1 penalty). As a result, for high values of , many coefficients are exactly zeroed under lasso, which is never the case in ridge regression.

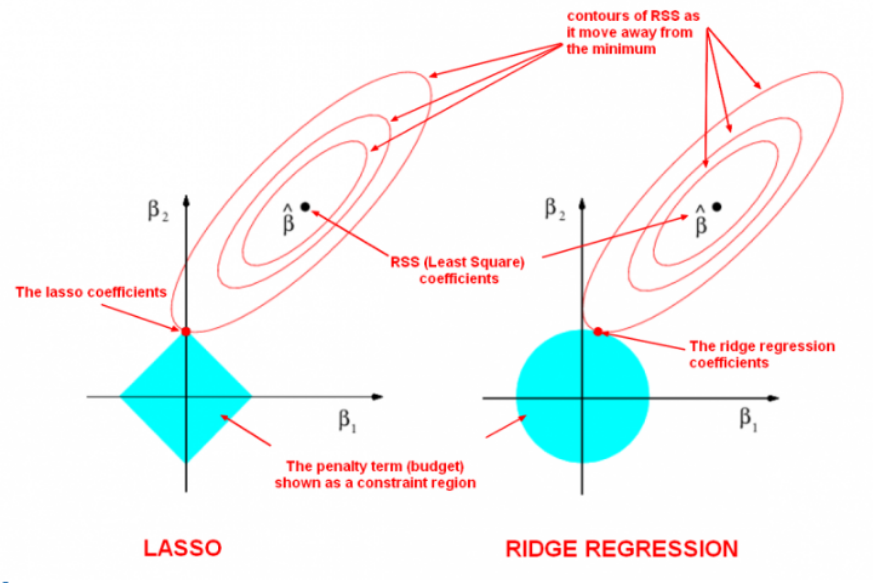


Figure 12: lasso vs ridge regression

Model specification: The only difference in ridge and lasso loss functions is in the penalty terms. Under lasso, the loss is defined as:

$$L_{lasso}(\hat{\beta}) = \sum_{i=1}^{n}(y_i - x_i'\hat{\beta}) + \lambda \sum_{i=1}^{n} |\hat{\beta}| \tag{6}$$

Major details are given in the paper of Tibshirani(1996) , or read more here

In our study, we used the **glmnet** functions of the glmnet package to fit our lasso regression model. Then , we predicted our fitted model on test set. **glmnet** solves the following problem

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1\right]$$

, over a grid of values of covering the entire range. Here $l(y,)$ is the negative log-likelihood contribution for observation i; e.g. for the Gaussian case it is $\frac{1}{2}(y)^2$. The elastic-net penalty is controlled by , and bridges the gap between lasso (= 1, the default) and ridge (= 0). The tuning parameter controls the overall strength of the penalty.

4.2 Model performance evaluation

A classification model gives a predicted class for every case in the test data set, the actual class of each of them is known. The confusion matrix provides a convenient way to compare the frequencies of actual versus predicted class for the test data set. Two-class problem is the most common situation in credit scoring. The credit customers are usually classified into two classes: 'good' and 'bad' (or 'accepted' and'rejected', 'default' and 'non-default')(Liu,2002). The format of confusion matrix for two-class problem is shown in the following table from Wang et al (2011).

Confusion matrix for credit scoring.

		Actual condition	
		Positive (Non-Risk)	Negative (Risk)
Test result	Positive(Non-Risk)	True Positive (TP)	False Positive (FP)
	Negative (Risk)	False Negative (FN)	True Negative (TN)

where

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{7}$$

$$TPR(sensitivy\ , or\ true\ positive\ rate) = \frac{TP}{TP + FN} = 1 - FNR \tag{8}$$

where FNR(false negative rate is Type I error) , $FNR = \frac{FN}{FN+TP}$

$$TNR(specificity\, or\ true\ negative\ rate) = \frac{TN}{TN + FP} = 1 - FPR \tag{9}$$

where FPR(false positive rate is Type II error) , $FPR = \frac{FP}{FP+TN}$

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

Moreover, always following Liu(2002), we defined another evaluation criteria: **ROC curve**. In statistics, a receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. A measure is given by the area under the ROC curve (denoted as **AUC**). The curve that has a larger AUC is better than the one that has a smaller AUC (Bock,2001). As in Hand & Till (2001) ,it can be show that the AUC is related to the Gini coefficient (G_1) by the formula $G_1 = 2\text{AUC} - 1 G_1 = 2\text{AUC} - 1$, where:

$$G_1 = 1 - \sum_{k=1}^{n} (X_k - X_{k-1})(Y_k + Y_{k-1}) \tag{12}$$

In our study, for each predicted model on test set,we used the **confusionMatrix() function** of the caret package to get values of Accuracy, sensitivity and specicifity and other metrics . We plotted ROC curves and Recall/precision curves using the ROCR package.

5 Results

5.1 Identifying the best model

5.1.1 Logistic Regression Models

The summary output of fitted **reg_model** shows that attributes such as check_acct, credit_hist, Purpose , Amount, ForeignWorkers, present_emp or installment_rate are strongly statistically significant with a p-value not only less than 0.05, but also 0.01. The Akaike information criterion (AIC), that is negative log likelihood penalized for a number of parameters, indicates a value equals to 693.24. Lower value of AIC suggests "better" model, but it is a relative measure of model fit. It is used for model selection, i.e. it lets you to compare different models estimated on the same dataset. AIC is useful for comparing models, but it does not tell you anything about the goodness of fit of a single, isolated model. Read more here

```
"reg_model"
```

```
Call:
glm(formula = Risk ~ ., family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6584  -0.6519   0.3478   0.6902   2.2646

Coefficients:
                                Estimate Std. Error z value Pr(>|z|)
(Intercept)                   -2.117e-01  1.091e+00  -0.194 0.846121
check_acct0.to.200             3.815e-01  2.708e-01   1.409 0.158861
check_acctgt.200               8.285e-01  4.153e-01   1.995 0.046079 *
check_acctnone                 1.885e+00  2.880e-01   6.543 6.03e-11 ***
credit_histThisBank.AllPaid   -2.089e-01  6.264e-01  -0.334 0.738718
credit_histPaidDuly            8.487e-01  4.681e-01   1.813 0.069833 .
credit_histDelay               1.861e+00  5.900e-01   3.154 0.001609 **
credit_histCritical            1.465e+00  4.971e-01   2.946 0.003220 **
Duration                      -2.286e-02  1.096e-02  -2.085 0.037098 *
savings_acct100.to.500        -1.165e-01  3.600e-01  -0.324 0.746155
savings_acct500.to.1000        5.297e-01  5.111e-01   1.036 0.299982
savings_acctgt.1000            1.307e+00  6.388e-01   2.047 0.040677 *
savings_acctUnknown            6.274e-01  3.116e-01   2.014 0.044045 *
PurposeUsedCar                 2.087e+00  4.611e-01   4.527 5.99e-06 ***
PurposeFurniture.Equipment     1.124e+00  3.234e-01   3.476 0.000508 ***
PurposeRadio.Television        8.793e-01  3.025e-01   2.907 0.003653 **
PurposeDomesticAppliance       6.807e-01  9.890e-01   0.688 0.491295
PurposeRepairs                -4.738e-01  6.437e-01  -0.736 0.461716
PurposeEducation               1.398e-01  4.636e-01   0.302 0.763003
PurposeRetraining              2.168e+00  1.229e+00   1.764 0.077771 .
PurposeBusiness                1.162e+00  4.244e-01   2.738 0.006186 **
PurposeOther                   1.441e+00  8.382e-01   1.720 0.085511 .
Age                            1.764e-02  1.107e-02   1.594 0.110991
PropertyInsurance             -3.013e-01  3.070e-01  -0.982 0.326234
PropertyCarOther               1.288e-02  2.795e-01   0.046 0.963260
PropertyUnknown               -5.114e-01  5.442e-01  -0.940 0.347397
Amount                        -1.540e-04  5.046e-05  -3.052 0.002271 **
present_emp1.to.4              1.087e-01  2.969e-01   0.366 0.714383
present_emp4.to.7              1.177e+00  3.887e-01   3.029 0.002453 **
present_empgt.7                3.926e-01  3.588e-01   1.094 0.273852
present_empUnemployed          1.227e-01  4.705e-01   0.261 0.794345
HousingOwn                     4.664e-01  2.728e-01   1.710 0.087323 .
HousingForFree                 5.462e-01  5.989e-01   0.912 0.361745
other_installStores            2.697e-01  4.940e-01   0.546 0.585065
other_installNone              6.898e-01  2.931e-01   2.354 0.018589 *
status_sexFemale.NotSingle     4.855e-01  4.502e-01   1.078 0.280838
status_sexMale.Single          8.701e-01  4.394e-01   1.980 0.047690 *
status_sexMale.Married.Widowed 4.273e-01  5.258e-01   0.813 0.416434
ForeignWorkeryes              -2.082e+00  7.551e-01  -2.758 0.005821 **
other_debtCoApplicant         -7.713e-01  5.125e-01  -1.505 0.132291
other_debtGuarantor            1.213e+00  5.359e-01   2.263 0.023606 *
installment_rate              -3.064e-01  1.067e-01  -2.872 0.004085 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 855.21  on 699  degrees of freedom
Residual deviance: 609.24  on 658  degrees of freedom
AIC: 693.24

Number of Fisher Scoring iterations: 5


 "reg_model: prediction on test set and build confusionMatrix"


Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  44  27
         1  46 183

               Accuracy : 0.7567
                 95% CI : (0.704, 0.8041)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.01741

                  Kappa : 0.3834
 Mcnemar's Test P-Value : 0.03514

            Sensitivity : 0.4889
            Specificity : 0.8714
         Pos Pred Value : 0.6197
         Neg Pred Value : 0.7991
             Prevalence : 0.3000
         Detection Rate : 0.1467
   Detection Prevalence : 0.2367
      Balanced Accuracy : 0.6802

       'Positive' Class : 0
```

We observed that top 5 important predictors according to the glm model are: check_acct ,Purpose , credit_hist , Amount and present_emp. We also noticed how these predictors are part of the list of statistically significant attributes (see *summary(reg_model)*) . We used them to fit a new logistic regression model based on the same training set.

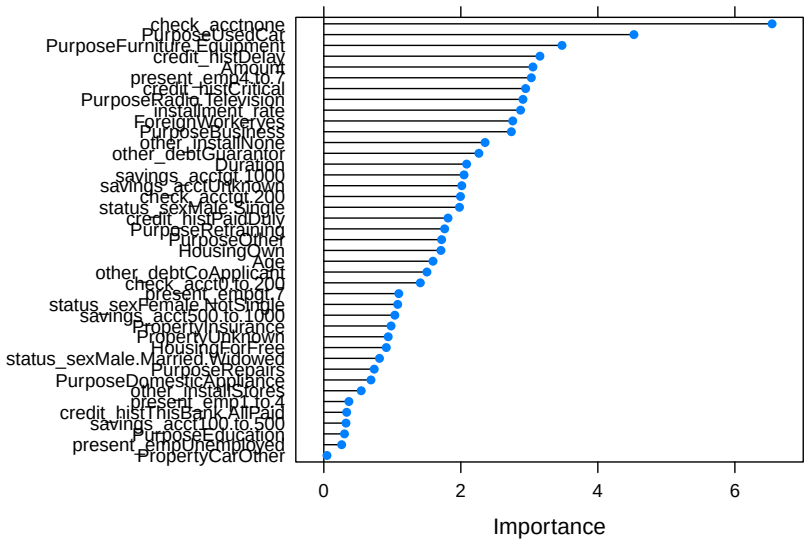


Figure 13: variable importance- logistic model

```
 "reg_model.new"


Call:
glm(formula = Risk ~ check_acct + Purpose + credit_hist + Amount +
    present_emp, family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5463  -0.8206   0.4263   0.7722   2.1471

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                 -1.400e+00  5.356e-01  -2.615 0.008930 **
check_acct0.to.200           4.876e-01  2.385e-01   2.045 0.040898 *
check_acctgt.200             1.091e+00  3.862e-01   2.825 0.004733 **
check_acctnone               1.919e+00  2.615e-01   7.341 2.13e-13 ***
PurposeUsedCar               1.841e+00  4.321e-01   4.260 2.04e-05 ***
PurposeFurniture.Equipment   7.747e-01  2.907e-01   2.664 0.007712 **
PurposeRadio.Television      5.425e-01  2.683e-01   2.022 0.043173 *
PurposeDomesticAppliance     5.961e-01  9.202e-01   0.648 0.517100
PurposeRepairs              -4.907e-01  5.981e-01  -0.820 0.411946
PurposeEducation            -3.794e-01  4.239e-01  -0.895 0.370829
PurposeRetraining            1.784e+00  1.155e+00   1.545 0.122401
PurposeBusiness              5.662e-01  3.797e-01   1.491 0.135962
PurposeOther                 8.496e-01  7.965e-01   1.067 0.286097
credit_histThisBank.AllPaid -2.097e-01  5.697e-01  -0.368 0.712751
credit_histPaidDuly          9.276e-01  4.444e-01   2.087 0.036856 *
credit_histDelay             1.746e+00  5.597e-01   3.119 0.001813 **
credit_histCritical          1.632e+00  4.752e-01   3.434 0.000594 ***
Amount                      -1.692e-04  3.803e-05  -4.449 8.61e-06 ***
present_emp1.to.4            4.144e-01  2.633e-01   1.574 0.115578
```

```
present_emp4.to.7           1.250e+00  3.524e-01   3.547 0.000389 ***
present_empgt.7             6.436e-01  2.971e-01   2.167 0.030271 *
present_empUnemployed       2.192e-01  4.314e-01   0.508 0.611309
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 855.21  on 699  degrees of freedom
Residual deviance: 675.98  on 678  degrees of freedom
AIC: 719.98

Number of Fisher Scoring iterations: 5
```

"reg_model" seems to be a better model than "reg_model.new" , since this latter shows a AIC value equals to 719.98 .

```
 "reg_model.new: prediction on test set and build confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  30  25
         1  60 185

               Accuracy : 0.7167
                 95% CI : (0.662, 0.767)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.2873596

                  Kappa : 0.2411
 Mcnemar's Test P-Value : 0.0002262

            Sensitivity : 0.3333
            Specificity : 0.8810
         Pos Pred Value : 0.5455
         Neg Pred Value : 0.7551
             Prevalence : 0.3000
         Detection Rate : 0.1000
   Detection Prevalence : 0.1833
      Balanced Accuracy : 0.6071

       'Positive' Class : 0
```

Based on the overall accuracy values, reg_model (0.7567) is a more precise model than reg_model.new (0.7167). Let's define their performance to plot ROC curves , and calculate AUC values.

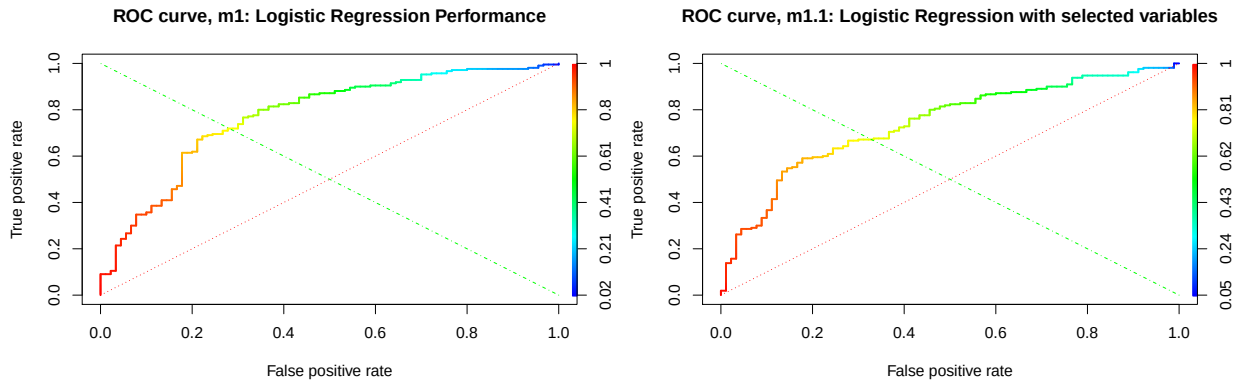


Figure 14: ROC curves-Logistic regression models

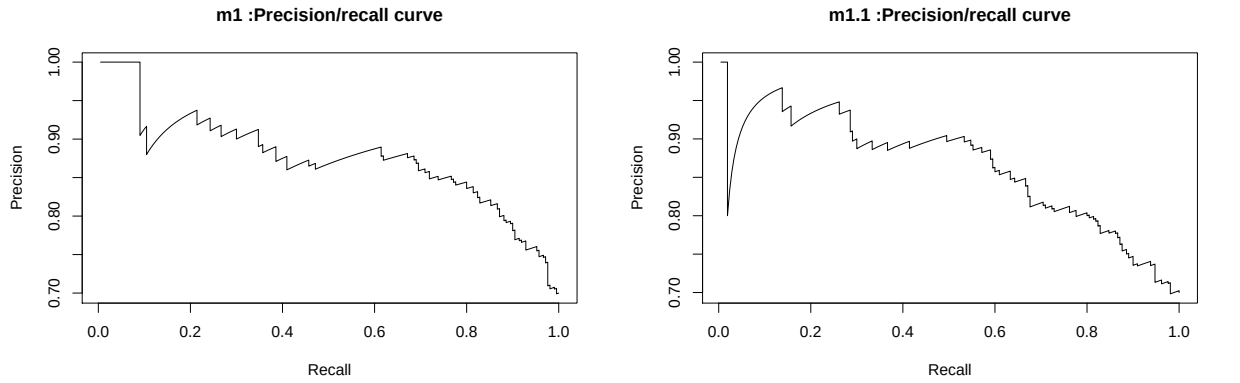


Figure 15: Precision/recall curves-Logistic regression models

In addition, we see that "reg_model" points out an AUC value greater than the "reg_model.new" one.

```
"AUC:reg_model"
```

```
0.7741
```

```
"AUC:reg_model.new"
```

```
0.7408
```

5.1.2 Decision trees

We observed that the tree fitted on the training set has has a root node, 13 splits and 14 leaves (terminal nodes). We can also see values of CP: Internally, rpart keeps track of something called the complexity of a tree. The complexity measure is a combination of the size of a tree and the ability of the tree to separate the classes of the target variable. If the next best split in growing a tree does not reduce the tree's overall complexity by a certain amount, rpart will terminate the growing process.

The rel error of each iteration of the tree is the fraction of mislabeled elements in the iteration relative to the fraction of mislabeled elements in the root. In this example, 30% of training cases have bad credit risk (see root node error). The cross validation error rates and standard deviations are displayed in the columns xerror and xstd respectively. According to our rpart plots, Our fitted model suggests that Check_acct is the best attribute to classify credit worthiness of applicants.

```
 "we display CP table for Fitted Rpart Object : dt_model"

 Age          Amount       check_acct   credit_hist  Duration
[6] other_debt   present_emp  Purpose      savings_acct

Root node error: 210/700 = 0.3

n= 700

        CP nsplit rel error  xerror     xstd
1 0.050794      0   1.00000 1.00000 0.057735
2 0.014286      4   0.78095 0.93333 0.056569
3 0.012698      8   0.71905 0.94286 0.056744
4 0.010000     14   0.63333 0.93810 0.056656
```

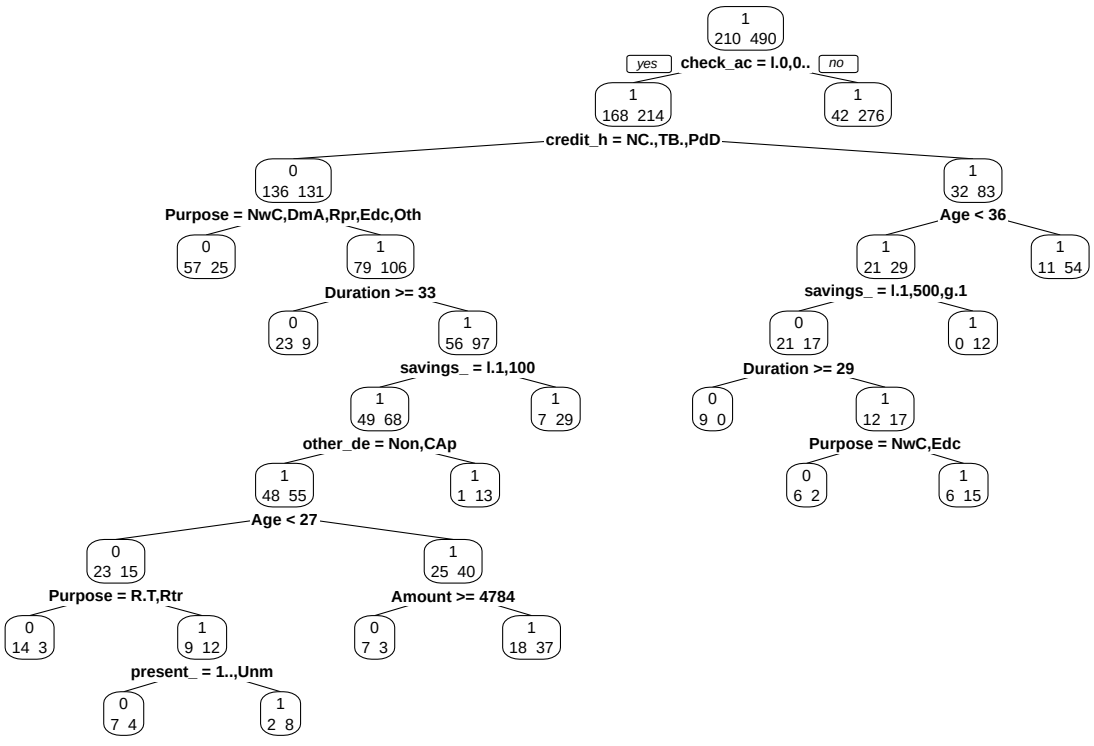


Figure 16: rpart plot of decision tree model

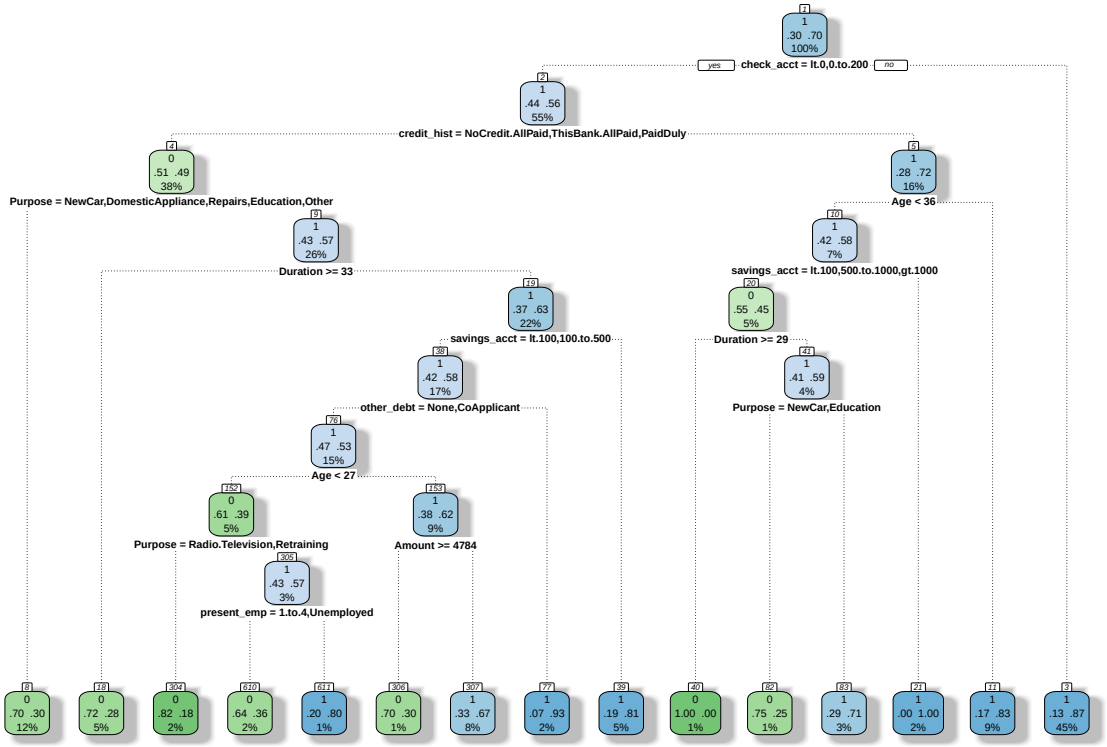


Figure 17: fancyrpart plot of decision tree model

```
 "dt_model:prediction on test set and build confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  43  36
         1  47 174

               Accuracy : 0.7233
                 95% CI : (0.669, 0.7732)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.2072

                  Kappa : 0.3174
 Mcnemar's Test P-Value : 0.2724

            Sensitivity : 0.4778
            Specificity : 0.8286
         Pos Pred Value : 0.5443
         Neg Pred Value : 0.7873
             Prevalence : 0.3000
         Detection Rate : 0.1433
```

```
   Detection Prevalence : 0.2633
      Balanced Accuracy : 0.6532

       'Positive' Class : 0
```

Prediction on unseen data performed enough well with an overall accuracy of about 0.72 . However, it's usually a good idea to prune a decision tree. Fully grown trees don't perform well against data not in the training set because they tend to be over-fitted so pruning is used to reduce their complexity by keeping only the most important splits.

```
 "prune by lowest cp - and output length of pruned tree"

 5

 "Displays CP table for Fitted - pruned tree"

 check_acct  credit_hist Duration    Purpose

Root node error: 210/700 = 0.3

n= 700

        CP nsplit rel error  xerror     xstd
1 0.050794      0   1.00000 1.00000 0.057735
2 0.014286      4   0.78095 0.93333 0.056569

 "count the number of splits"

 4
```

From the rpart documentation, "An overall measure of variable importance is the sum of the goodness of split measures for each split for which it was the primary variable" .When rpart grows a tree it performs 10-fold cross validation on the data. The output of **printcp(dt_model)** show us a list of cp values. We selected the one having the least cross-validated error and used it to prune the tree, since the value of cp should be least, so that the cross-validated error rate is minimum. This corresponds to CP equals to 0.01. As we can observe, the pruned tree only keeps the 4 most important splits.

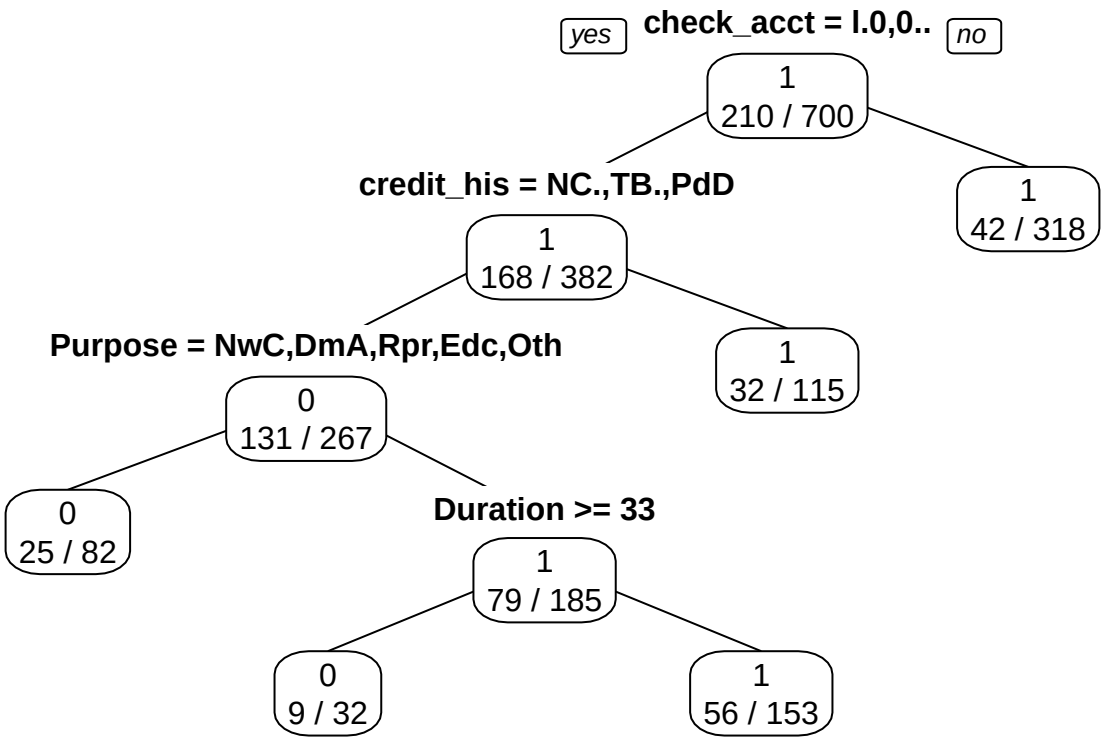


Figure 18: rpart plot of pruned tree

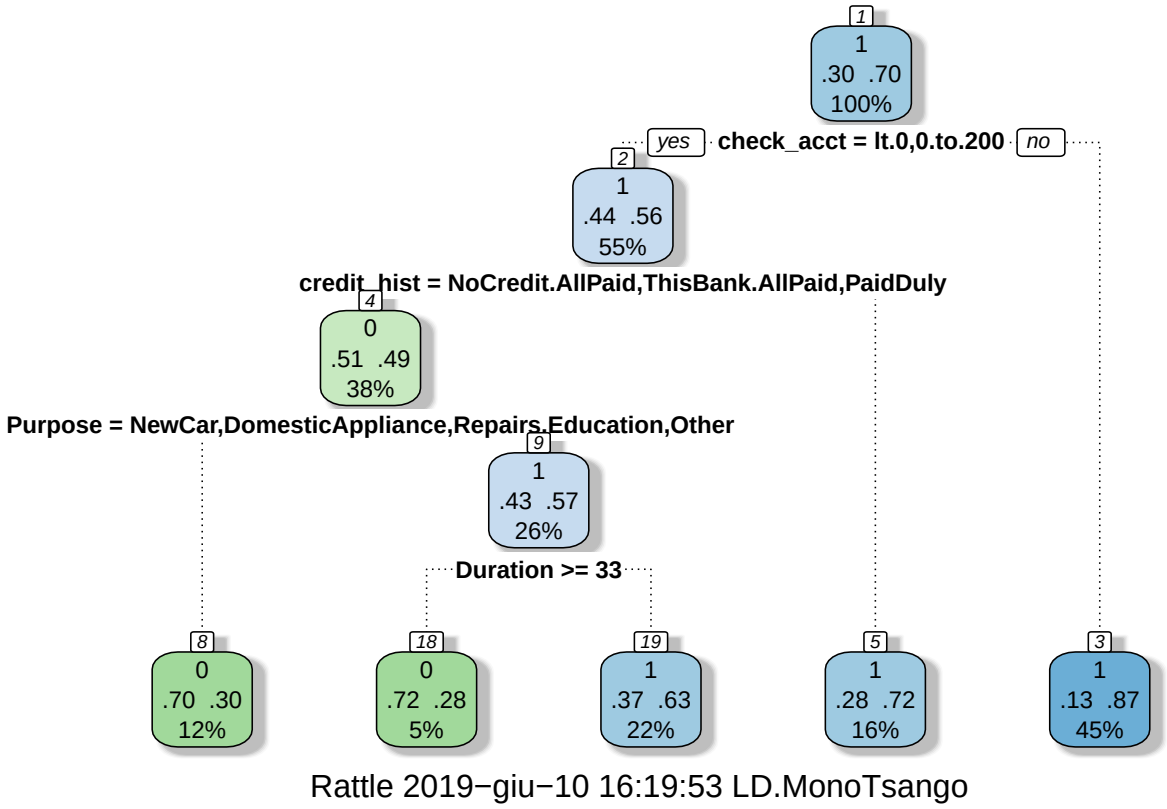


Figure 19: fancyrpart plot of pruned tree

This is the situation after pruning : We observe how the tree keeps the important splits : a root node, 4 splits, and 5 terminal nodes. The first splitting criteria is " check_acct = lt.0, 0.to.200", which separates the data into a set of 45 cases (which 39 have good credit worthiness and 6 , a bad one) and a set of 55 cases (which 31 have a good credit risk and 24 a bad credit risk) . The latter have a further split based on criteria "Credit_hist : Notcredit.Allpaid, ThisBank.Allpaid, PaidDuly" " into two sets :a set of 16 cases (which 11 have a good credit risk, and 5 not) , and a set of 38 cases (which 18 have a good credit risk, and almost 20 a bad one) . Again, this latter have a further split based on criteria "Purpose: NewCar, Domesticappliance,Repairs,Education,Other" where the 38 cases are separated into 26 (which 15 have a good credit profile and 11 a bad one) cases and 12 cases(which 4 have a good credit risk and 8 a bad credit risk). The last splitting criteria based on "Duration > 33" separates these 26 cases (label9) into labels 18 and 19.

```
 "pruned.tree : prediction on test set and  confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  33  26
         1  57 184

               Accuracy : 0.7233
                 95% CI : (0.669, 0.7732)
```

```
   No Information Rate : 0.7
   P-Value [Acc > NIR] : 0.2072197

                 Kappa : 0.2694
Mcnemar's Test P-Value : 0.0009915

            Sensitivity : 0.3667
            Specificity : 0.8762
         Pos Pred Value : 0.5593
         Neg Pred Value : 0.7635
             Prevalence : 0.3000
         Detection Rate : 0.1100
   Detection Prevalence : 0.1967
      Balanced Accuracy : 0.6214

       'Positive' Class : 0
```

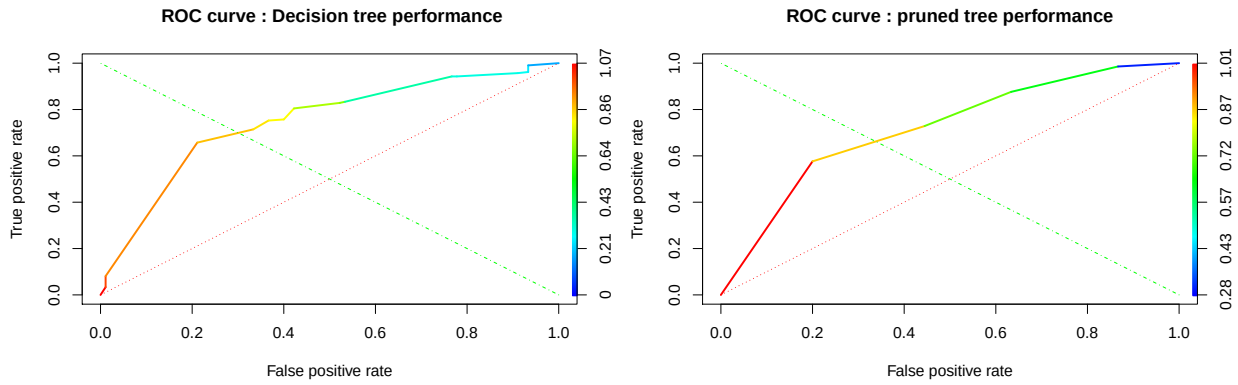


Figure 20: ROC curves-Decision trees

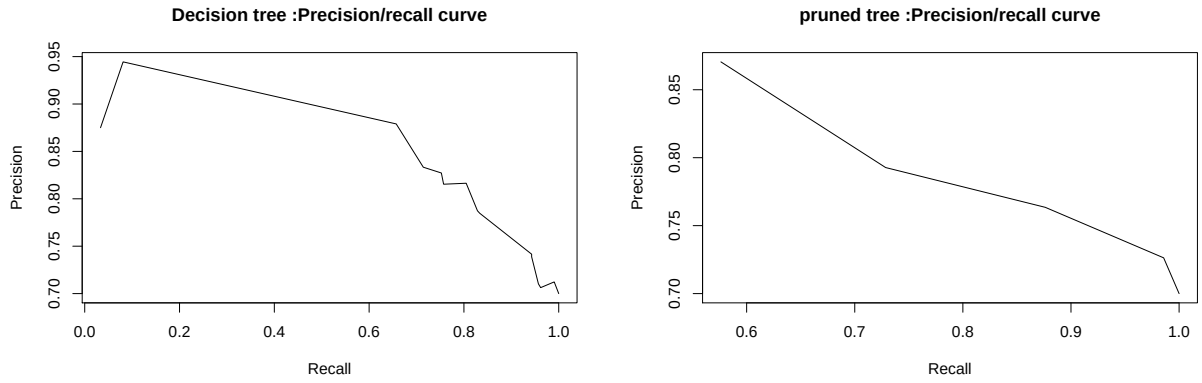


Figure 21: Precision/recall curves-Decision trees

```
"AUC:dt_model"
```

```
 0.7477
```

```
 "AUC:dt2_model(pruned.tree)"
```

```
 0.7183
```

The two decision tree models seem to be equally accurate, since they have the same overall accuracy value. The discrimination through the balanced accuracy and AUC values help us to point out that "dt_model" is better than "pruned.tree" .

5.1.3 random forests

Number of trees is 500 and number of variables tried at each split is 3 in this case. The nature of a random forest is it iteratively uses a different subset of the data (bootstrap aggregation) to make multiple decision trees. At each iteration, the tree created using the subset is tested with the data that is not used to create the tree. The average of errors of all these interactions is the Out of Bag Error or misclassification rate, in this case equals to 25.14% .

```
 "rf_model : print fitted random forest model"
```

```
Call:
 randomForest(formula = Risk ~ ., data = train, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 25.14%
Confusion matrix:
   0   1 class.error
0 75 135  0.64285714
1 41 449  0.08367347
```

Let's make prediction on test set.

```
 "rf_model: prediction on test set and confusionMatrix"
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  37  13
         1  53 197

               Accuracy : 0.78
                 95% CI : (0.7288, 0.8256)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.001189

                  Kappa : 0.4
 Mcnemar's Test P-Value : 1.582e-06
```

```
            Sensitivity : 0.4111
            Specificity : 0.9381
         Pos Pred Value : 0.7400
         Neg Pred Value : 0.7880
             Prevalence : 0.3000
         Detection Rate : 0.1233
   Detection Prevalence : 0.1667
      Balanced Accuracy : 0.6746

       'Positive' Class : 0
```

We tried to improve our random forest model with the important features through variable importance based on the MeanDecrease Gini.

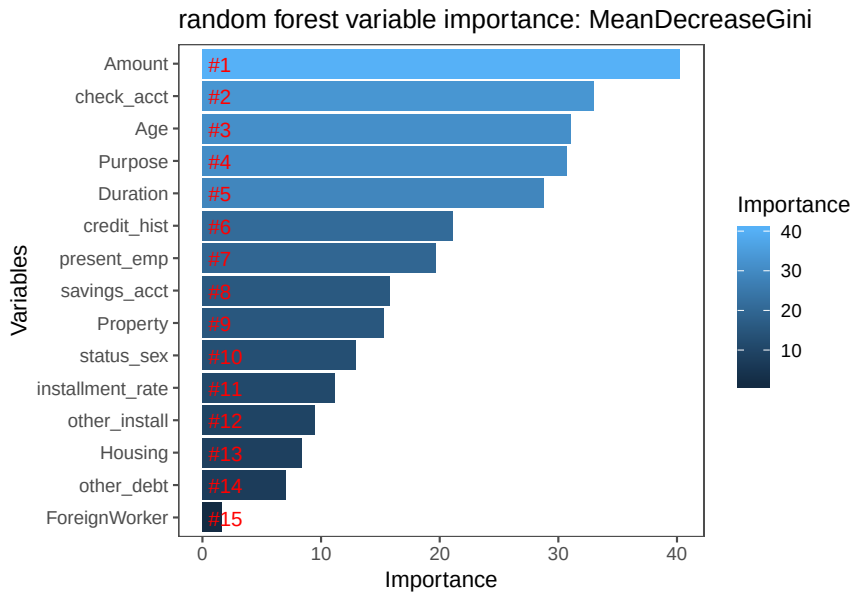


Figure 22: random forest variable importance

we built a new rf model (rf_model.new) with the 5 top variables based on MeanDecrease Gini

```
 "rf_model.new : print fitted random forest model"


Call:
 randomForest(formula = Risk ~ Amount + check_acct + Age + Purpose +      Duration, data = train, impor
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 25.86%
Confusion matrix:
   0   1 class.error
0 94 116   0.5523810
1 65 425   0.1326531
```

```
 "rf_model.new: prediction on test set and confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  41  23
         1  49 187

               Accuracy : 0.76
                 95% CI : (0.7076, 0.8072)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.012488

                  Kappa : 0.3772
 Mcnemar's Test P-Value : 0.003216

            Sensitivity : 0.4556
            Specificity : 0.8905
         Pos Pred Value : 0.6406
         Neg Pred Value : 0.7924
             Prevalence : 0.3000
         Detection Rate : 0.1367
   Detection Prevalence : 0.2133
      Balanced Accuracy : 0.6730

       'Positive' Class : 0
```

Now we are going to tune/find the best parameters with 10-fold cross validation to build optimal random forest model

```
 "print tuning.results"


Parameter tuning of 'randomForest':

- sampling method: 10-fold cross validation

- best parameters:
 nodesize mtry ntree
        4    3   500

- best performance: 0.2357143

 "rf_model.best: prediction on test set and confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  38  13
```

```
         1  52 197

               Accuracy : 0.7833
                 95% CI : (0.7324, 0.8286)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.0007553

                  Kappa : 0.4112
 Mcnemar's Test P-Value : 2.437e-06

            Sensitivity : 0.4222
            Specificity : 0.9381
         Pos Pred Value : 0.7451
         Neg Pred Value : 0.7912
             Prevalence : 0.3000
         Detection Rate : 0.1267
   Detection Prevalence : 0.1700
      Balanced Accuracy : 0.6802

       'Positive' Class : 0
```

Based on their overall accuracy values, we plot the ROC curves of potential best two models. (rf_model & rf_model.best)

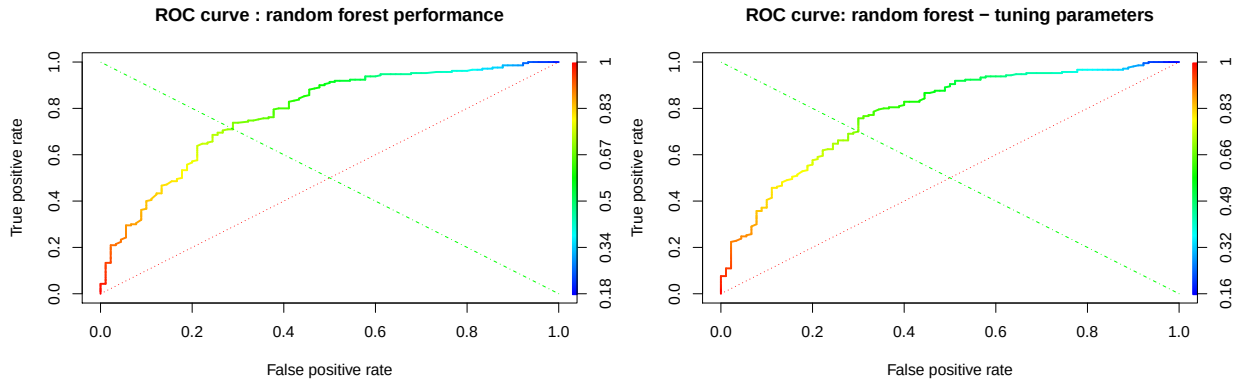


Figure 23: ROC curves-random forests

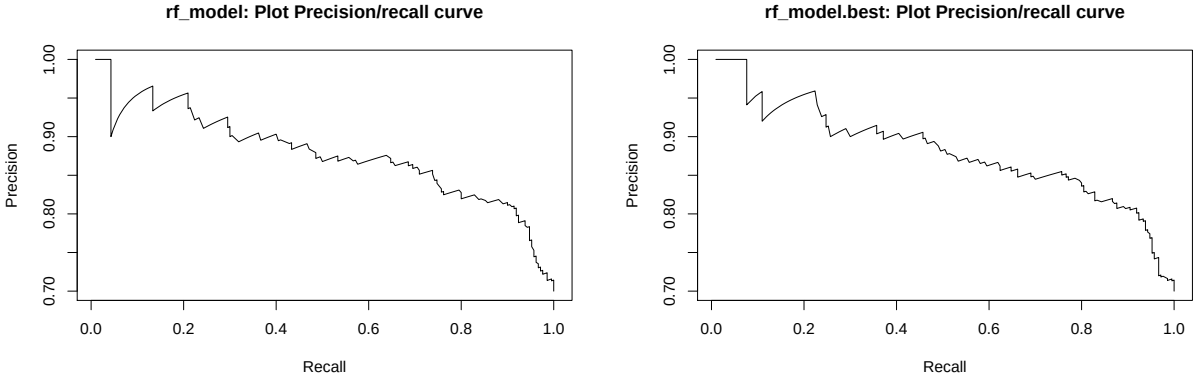


Figure 24: Precision/recall curves-random forests

```
"AUC:rf_model"

0.7805

"AUC:rf_model.new"

0.749

"AUC:rf_model.best"

0.7813
```

The optimized random forest model which shows that best parameters are nodesize = 4 , mtry = 3 , ntree = 500 not only indicates a higher overall accuracy than other random forest models, but also a greater AUC value.

Decision tree is encountered with over-fitting problem and ignorance of a variable in case of small sample size and large p-value. Whereas, random forests are a type of recursive partitioning method particularly well-suited to small sample size and large p-value problems.Random forest comes at the expense of a some loss of interpretability, but generally greatly boosts the performance of the final model.

5.1.4 Support Vector Machines

C controls the cost of misclassification on the training data. Small C makes the cost of misclassificaiton low ("soft margin"), thus allowing more of them for the sake of wider "cushion". Large C makes the cost of misclassification high ('hard margin"), thus forcing the algorithm to explain the input data stricter and potentially overfit. Usually a very high number of support vectors like the one we obtained, 699, indicates you are overfitting. This presumably caused by the big value of C (C=100) we used.

```
 "svm_model:print fitted model "


Call:
svm(formula = Risk ~ ., data = train, kernel = "radial", gamma = 1,
    cost = 100)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  100
      gamma:  1

Number of Support Vectors:  699

 "svm_model:prediction on test set and confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   1   2
         1  89 208

               Accuracy : 0.6967
                 95% CI : (0.6412, 0.7482)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.5781

                  Kappa : 0.0022
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.011111
            Specificity : 0.990476
         Pos Pred Value : 0.333333
         Neg Pred Value : 0.700337
             Prevalence : 0.300000
         Detection Rate : 0.003333
   Detection Prevalence : 0.010000
      Balanced Accuracy : 0.500794

       'Positive' Class : 0
```

We looked to improve our first svm model by finding the top 5 important variables and then adding effects of best tuning parameters of cost and gamma. Then , we used the ksvm function of kernlab package to build a svm model based on vanilladot Linear kernel function.

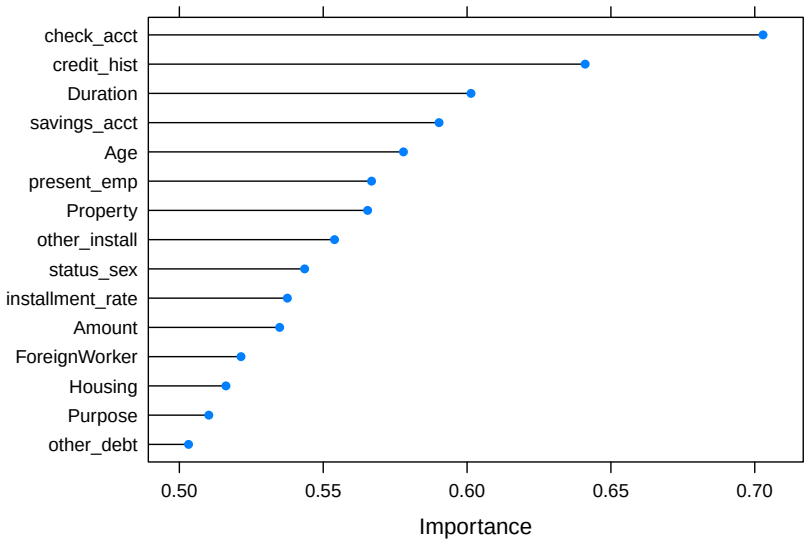


Figure 25: important features selected by 10 fold cv

```
 "print tuning.results"


Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
  100  0.01

- best performance: 0.2685714

 "svm_model.best:print fitted model "


Call:
best.tune(method = svm, train.x = Risk ~ check_acct + credit_hist +
    Duration + savings_acct + Age, data = train, ranges = list(cost = cost.weights,
    gamma = gamma.weights), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  100
      gamma:  0.01

Number of Support Vectors:  403

 "svm_model.best:prediction on test set and confusionMatrix"
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  26  12
         1  64 198

               Accuracy : 0.7467
                 95% CI : (0.6935, 0.7949)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.04285

                  Kappa : 0.2776
 Mcnemar's Test P-Value : 4.913e-09

            Sensitivity : 0.28889
            Specificity : 0.94286
         Pos Pred Value : 0.68421
         Neg Pred Value : 0.75573
             Prevalence : 0.30000
         Detection Rate : 0.08667
   Detection Prevalence : 0.12667
      Balanced Accuracy : 0.61587

       'Positive' Class : 0
```

"svm_model.best" shows a great improvement of overall accuracy value with respect to "svm_model"

```
 Setting default kernel parameters

 "svm_model.vanilla : print fitted model"

Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 374

Objective Function Value : -346.6312
Training error : 0.208571

 "svm_model.vanilla:prediction on test set and confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
```

```
         0  46  29
         1  44 181

               Accuracy : 0.7567
                 95% CI : (0.704, 0.8041)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.01741

                  Kappa : 0.3917
 Mcnemar's Test P-Value : 0.10130

            Sensitivity : 0.5111
            Specificity : 0.8619
         Pos Pred Value : 0.6133
         Neg Pred Value : 0.8044
             Prevalence : 0.3000
         Detection Rate : 0.1533
   Detection Prevalence : 0.2500
      Balanced Accuracy : 0.6865

       'Positive' Class : 0
```

Based on their overall accuracy values, we plot the ROC curves of potential best two models. (svm.best_model & svm_model.vanilla) .

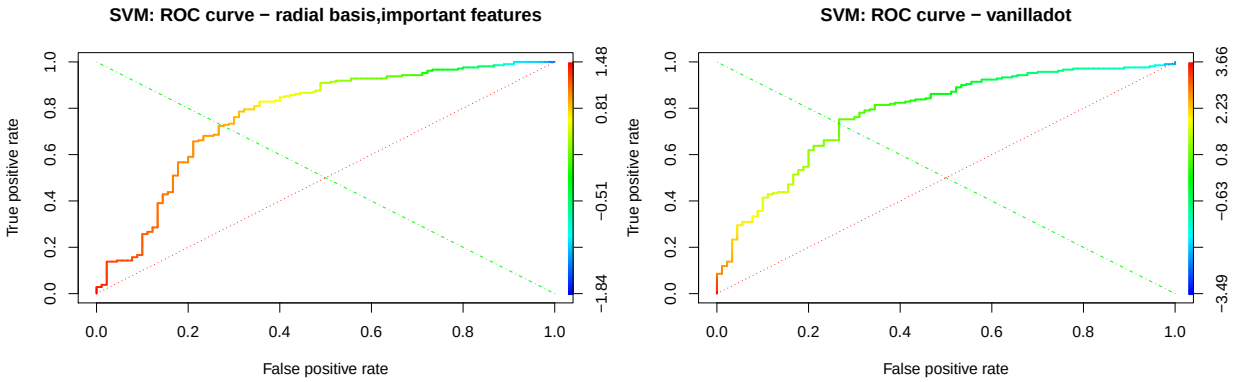


Figure 26: ROC curves-SVM

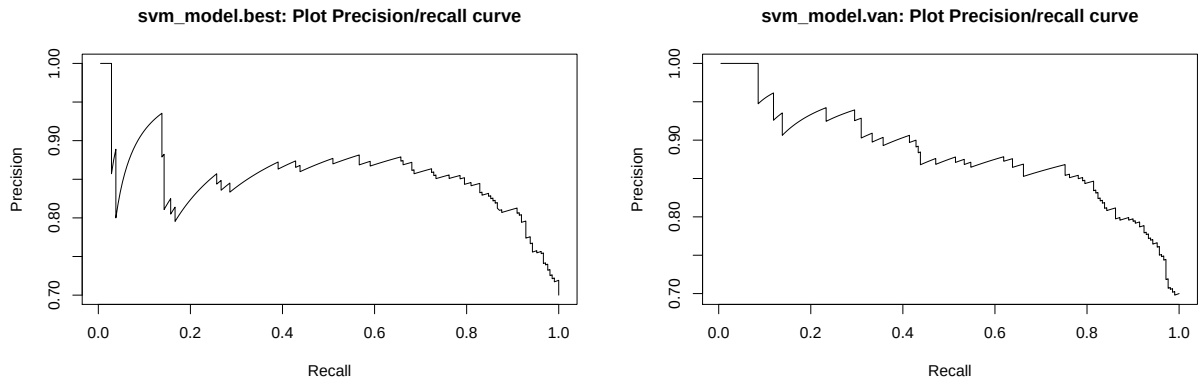


Figure 27: Precision/recall curves-random forests

```
"AUC:svm_model"

0.7313

"AUC:svm_model.best"

0.7692

"AUC:svm_model.vanilla"

0.7794
```

SVMs are great classifiers for specific situations when the groups are clearly separated. They also do a great job when your data is non-linearly separated. You could transform data to linearly separate it, or you can have SVMs convert the data and linearly separate the two classes right out of the box. This is one of the main reasons to use SVMs. You don't have to transform non-linear data yourself. One downside to SVMs is the black box nature of these functions. The use of kernels to separate non-linear data makes them difficult (if not impossible) to interpret. Understanding them will give you an alternative to GLMs and decision trees for classification.

5.1.5 Neural Networks

To perform neural network models we need numeric data .Normally, a neural network doesn't fit well with low number of observations. First avoid, we performed a specific Data pre-processing using the original data copy "**german.copy**" instead of "**german.copy2**" to create a new dataset copy, "**german.copy3**" . For this new recoding of "german.copy3", we kept the same attributes ($IV > 2$) and the numerical target "*risk_bin*" . The Data splitting is made as before with the **createDataPartition()** function to correct imbalanced class. (see details in rmd report)

Why neural network needs normalization/standardization ? : If the input variables are combined linearly, as in an MLP, then it is rarely strictly necessary to standardize the inputs, at least in theory. The reason is that any rescaling of an input vector can be effectively undone by changing the corresponding weights and biases, leaving you with the exact same outputs as you had before. However, there are a variety of practical reasons why standardizing the inputs can make training faster and reduce the chances of getting stuck in local optima. Also, weight decay and Bayesian estimation can be done more conveniently with standardized

inputs. Standardizing inputs variables tends to make the training process better behaved by improving the numerical condition (see ftp://ftp.sas.com/pub/neural/illcond/illcond.html) of the optimization problem and ensuring that various default values involved in initialization and termination are appropriate.

read more here or at this link.

```
 "Data pre-processing for Neural network models"
```

```
 "german.copy3"
```

```
Observations: 1,000
Variables: 16
$ check_acct       <dbl> 1, 2, 4, 1, 1, 4, 4, 2, 4, 2, 2, 1, 2, 1, 1, ...
$ credit_hist      <dbl> 5, 3, 5, 3, 4, 3, 3, 3, 3, 5, 3, 3, 3, 5, 3, ...
$ Duration         <dbl> 6, 48, 12, 42, 24, 36, 24, 36, 12, 30, 12, 48...
$ savings_acct     <dbl> 5, 1, 1, 1, 1, 5, 3, 1, 4, 1, 1, 1, 1, 1, 1, ...
$ Purpose          <dbl> 4, 4, 7, 3, 1, 7, 3, 2, 4, 1, 1, 10, 4, 1, 1,...
$ Age              <dbl> 67, 22, 49, 45, 53, 35, 53, 35, 61, 28, 25, 2...
$ Property         <dbl> 1, 1, 1, 2, 4, 4, 2, 3, 1, 3, 3, 2, 3, 3, 3, ...
$ Amount           <dbl> 1169, 5951, 2096, 7882, 4870, 9055, 2835, 694...
$ present_emp      <dbl> 4, 2, 3, 3, 2, 2, 4, 2, 3, 5, 1, 1, 2, 4, 2, ...
$ Housing          <dbl> 2, 2, 2, 3, 3, 3, 2, 1, 2, 2, 1, 1, 2, 2, 1, ...
$ other_install    <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
$ status_sex       <dbl> 3, 2, 3, 3, 3, 3, 3, 3, 1, 4, 2, 2, 2, 3, 2, ...
$ ForeignWorker    <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
$ other_debt       <dbl> 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ installment_rate <dbl> 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 3, 1, 4, 2, ...
$ risk_bin         <dbl> 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, ...
```

```
 "training set"
```

```
Observations: 700
Variables: 16
$ check_acct       <dbl> 0.0000000, 0.3333333, 0.0000000, 0.0000000, 1...
$ credit_hist      <dbl> 1.00, 0.50, 0.50, 0.75, 0.50, 0.50, 0.50, 1.0...
$ Duration         <dbl> 0.02941176, 0.64705882, 0.55882353, 0.2941176...
$ savings_acct     <dbl> 1.00, 0.00, 0.00, 0.00, 0.50, 0.00, 0.00, 0.0...
$ Purpose          <dbl> 0.3, 0.3, 0.2, 0.0, 0.2, 0.0, 0.9, 0.0, 0.0, ...
$ Age              <dbl> 0.85714286, 0.05357143, 0.46428571, 0.6071428...
$ Property         <dbl> 0.0000000, 0.0000000, 0.3333333, 1.0000000, 0...
$ Amount           <dbl> 0.050566744, 0.313689887, 0.419940574, 0.2542...
$ present_emp      <dbl> 0.75, 0.25, 0.50, 0.25, 0.75, 0.00, 0.00, 0.7...
$ Housing          <dbl> 0.5, 0.5, 1.0, 1.0, 0.5, 0.0, 0.0, 0.5, 0.0, ...
$ other_install    <dbl> 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
$ status_sex       <dbl> 0.6666667, 0.3333333, 0.6666667, 0.6666667, 0...
$ ForeignWorker    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, ...
$ other_debt       <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ installment_rate <dbl> 1.0000000, 0.3333333, 0.3333333, 0.6666667, 0...
$ risk_bin         <fct> 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, ...
```

```
 "test set"
```

```
Observations: 300
Variables: 16
$ check_acct       <dbl> 1.0000000, 1.0000000, 0.3333333, 1.0000000, 0...
$ credit_hist      <dbl> 1.00, 0.50, 0.50, 0.50, 1.00, 0.50, 0.50, 1.0...
$ Duration         <dbl> 0.11111111, 0.55555556, 0.55555556, 0.1111111...
$ savings_acct     <dbl> 0.00, 1.00, 0.00, 0.75, 0.00, 0.00, 0.25, 1.0...
$ Purpose          <dbl> 0.6, 0.6, 0.1, 0.3, 0.0, 0.3, 0.3, 0.3, 0.9, ...
$ Age              <dbl> 0.53703704, 0.27777778, 0.27777778, 0.7592592...
$ Property         <dbl> 0.0000000, 1.0000000, 0.6666667, 0.0000000, 0...
$ Amount           <dbl> 0.114647189, 0.568475284, 0.431068214, 0.1774...
$ present_emp      <dbl> 0.50, 0.25, 0.25, 0.50, 1.00, 0.25, 0.25, 0.7...
$ Housing          <dbl> 0.5, 1.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, ...
$ other_install    <dbl> 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, ...
$ status_sex       <dbl> 0.6666667, 0.6666667, 0.6666667, 0.0000000, 1...
$ ForeignWorker    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ other_debt       <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
$ installment_rate <dbl> 0.3333333, 0.3333333, 0.3333333, 0.3333333, 1...
$ risk_bin         <fct> 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, ...
```

Then, we fitted the neural network model with the nnet function.

```
 "neur.net : print fitted model"

a 15-20-1 network with 341 weights
inputs: check_acct credit_hist Duration savings_acct Purpose Age Property Amount present_emp Housing otl
output(s): risk_bin
options were - entropy fitting  decay=0.001

 "neur.net : prediction on test set and confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  41  57
         1  54 148

               Accuracy : 0.63
                 95% CI : (0.5726, 0.6848)
    No Information Rate : 0.6833
    P-Value [Acc > NIR] : 0.9786

                  Kappa : 0.1522
 Mcnemar's Test P-Value : 0.8494

            Sensitivity : 0.4316
            Specificity : 0.7220
         Pos Pred Value : 0.4184
         Neg Pred Value : 0.7327
             Prevalence : 0.3167
         Detection Rate : 0.1367
   Detection Prevalence : 0.3267
      Balanced Accuracy : 0.5768
```

```
'Positive' Class : 0
```

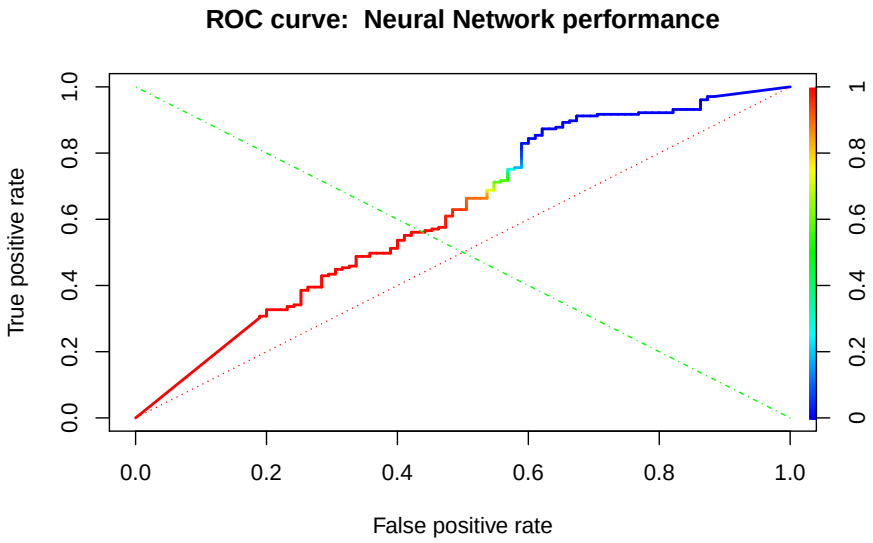


Figure 28: ROC curves-Neural Networks

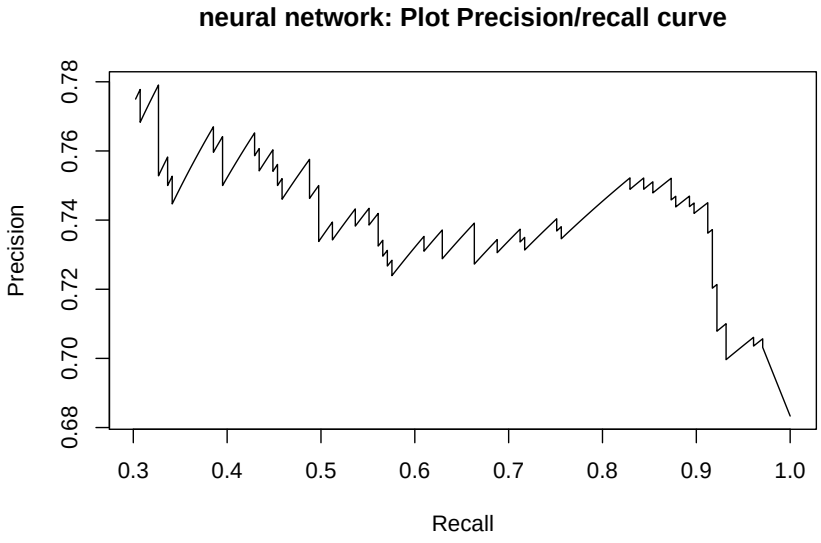


Figure 29: Precision/recall curves-Neural Networks

```
"AUC:neur.net model"
```

```
0.6185
```

Compared to the previous models, "neur.net" seems to be the less accurate model. It shows an overall accuracy of 0.63.

5.1.6 Lasso regression

For the lasso regression model, we used the same train_num and test_num we defined first for the neural network model. However, because the glmnet functions don't need a regression formula , but input matrices of predictors, we first constructed *mat_train* and *mat_test* with the **model.matrix** function. We fitted a GLM with lasso regularization, performing cross validation to find the optimal lambda ; alpha = 1 indicates that is LASSO and not elasticnet regression. We choose as loss measure to use for cross-validation "auc" , that is for two-class logistic regression only, and gives area under the ROC curve. (see codes in rmd report)

```
"lasso_model: print fitted model - see long output in rmd report"
```

The following plot shows the models (with different lambda values) that glmnet fitted, along with the auc values for each of the models.

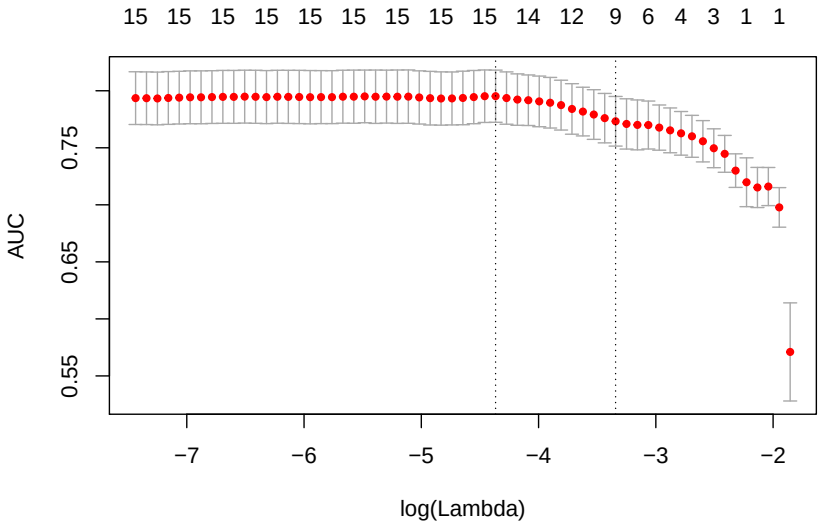


Figure 30: lasso model plot

It includes the cross-validation curve (red dotted line), and upper and lower standard deviation curves along the sequence (error bars). Two selected s are indicated by the vertical dotted lines (see below). We can view the selected s and the corresponding coefficients. For example, the minimum value of lambda , i.e the one that minimises the auc value corresponds to `lasso_model$lambda.min` . *lambda.min or lambda.1se ? :* The `lambda.min` option refers to value of at the lowest CV error. The error at this value of is the average of the errors over the k folds and hence this estimate of the error is uncertain. The `lambda.1se` represents the value of in the search that was simpler than the best model (lambda.min), but which has error within 1 standard error of the best model. In other words, using the value of lambda.1se as the selected value for results in a model that is slightly simpler than the best model but which cannot be distinguished from the best model in terms of error given the uncertainty in the k-fold CV estimate of the error of the best model. Read more here

In general though, the objective of regularization is to balance accuracy and simplicity. In the present context, this means a model with the smallest number of coefficients that also gives a good accuracy. To

this end, the cv.glmnet function finds the value of lambda that gives the simplest model but also lies within one standard error of the optimal value of lambda. This value of lambda(**lambda.1se**) is what we used to predict our fitted model on test set.

```
 "lasso_model : prediction on test set and confusionMatrix"

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  21   8
         1  74 197

               Accuracy : 0.7267
                 95% CI : (0.6725, 0.7763)
    No Information Rate : 0.6833
    P-Value [Acc > NIR] : 0.059

                  Kappa : 0.2237
 Mcnemar's Test P-Value : 7.071e-13

            Sensitivity : 0.22105
            Specificity : 0.96098
         Pos Pred Value : 0.72414
         Neg Pred Value : 0.72694
             Prevalence : 0.31667
         Detection Rate : 0.07000
   Detection Prevalence : 0.09667
      Balanced Accuracy : 0.59101

       'Positive' Class : 0
```

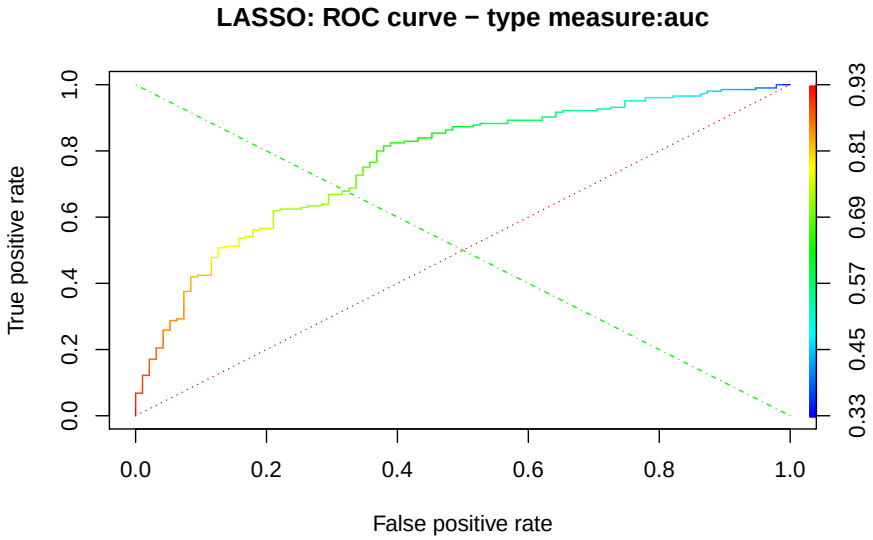


Figure 31: ROC curves-Lasso regression

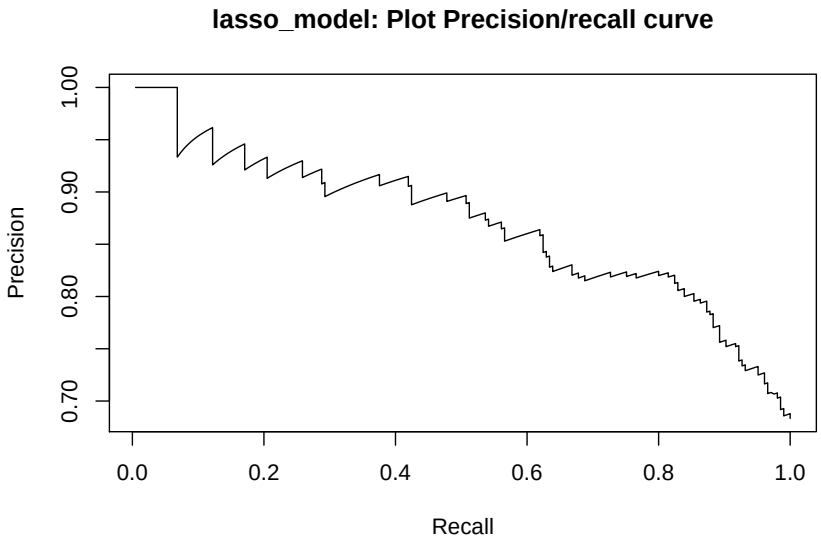


Figure 32: Precision/recall-Lasso regression

```
"AUC:lasso_model"
```

```
0.7675
```

As we can notice, Lasso regression model shows a good overall accuracy and good AUC performance.

5.2 Model optimization

Although we usually recommend studying both specificity and sensitivity, very often it is useful to have a one number summary, for example for optimization purposes. One metric that is preferred over overall accuracy is the average of specificity and sensitivity, referred to as balanced accuracy. Because specificity and sensitivity are rates, it is more appropriate to compute the harmonic average which give us the F1-score.Balanced accuracy is discussed in this paper, while further details for the F1-score are given here.

$$BalancedAccuracy\ (BACC) = \frac{\text{sensitivity} + \text{specificity}}{2}$$

$$F_1\ score = 2 \times \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

More over, we are going to define two other accuracy metrics **KS** and **Gini** that will support results of AUC. The kolmogorov-Smirnov test (KS-test) tries to determine if two classes differ significantly. In our case we are trying to measure whether the model is able to classify "good" class from "bad" class very well. If they are completely separated (i.e AUROC = 100%) then the value of KS will be 1(100%) and if they are same then the value of KS will be 0 (0%) . This means higher the value is better classification power . In simple term , *KS= Max distance between distributions of two classes* .

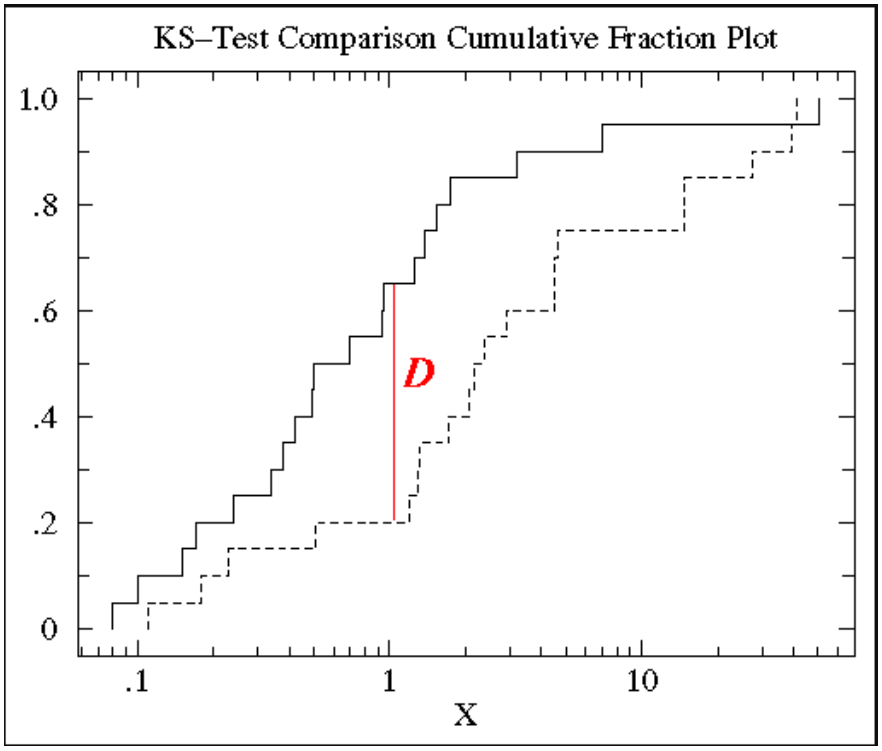


Figure 33: KS-Test comparison cumulative Fraction plot

Read more here : Massey(1951)

For the Gini coefficient , we already defined his relationship with the AUROC in paragraph 4.2 (*Gini = 2AUC -1*)

5.2.1 Model Performance Comparison - Measures of Accuracy

- **Balanced Accuracy , F1-score, Recall, Precision**

To get all these metric values, we can apply the formulas we defined in the previous paragraphs, or other way to get these different accuracy values is the following:

For the balanced accuracy, we can also get it directly from the output of the confusion matrix. `cm$byClass[['Balanced Accuracy']]` corresponds to the formula we define first. **((sensitivity + specificity) / 2)** . F1-score , The **F_meas** function in the caret package computes this summary with beta defaulting to 1 (Irrizary, 2018) .It corresponds to the F1 score formula we defined at the beginning of the paragraph. Also, We can notice that in paragraph IV.2, formula (8) is the same as formula (11). Then we can get recall metric as sensitivity from the output of the confusion matrix. Instead , for the Precision metric, formula(10) , its value in the confusionMatrix output corresponds to the PPV(positive pred value).Read more here

Models	Recall	Precision	BACC	F1-score	Accuracy
m1:Logistic regression	0.4889	0.6197	0.6802	0.5466	0.7567
m1_1:Logistic regression - important vars	0.3333	0.5455	0.6071	0.4138	0.7167
m2:Decision tree	0.4778	0.5443	0.6532	0.5089	0.7233
m2_1:Decision tree - pruning	0.3667	0.5593	0.6214	0.443	0.7233
m3:Random forest	0.4111	0.74	0.6746	0.5286	0.78
m3_1: Random forest - important vars	0.4556	0.6406	0.673	0.5325	0.76
m3_2: Random forest- tuning parameters	0.4222	0.7451	0.6802	0.539	0.7833
m4:SVM,radial	0.0111	0.3333	0.5008	0.0215	0.6967
m4_1:SVM,radial - important vars	0.2889	0.6842	0.6159	0.4062	0.7467
m4_2:SVM,vanilladot	0.5111	0.6133	0.6865	0.5576	0.7567
m5: Neural net	0.4316	0.4184	0.5768	0.4249	0.63
m6:Lasso reg	0.2211	0.7241	0.591	0.3387	0.7267

Table 1: Comparison of Model Performances- all fitted models

Here, we compared models by group, and based on *model_performance_metric_1* (see details and codes in the rmd report). The first criteria of comparison is the overall accuracy. Then, further discrimination is through F1-score, Balanced accuracy, precision and recall. For the first group , *Logistic regression models*, m1 is definitively better with all metrics greater than m1_1. For the second group, *decision trees*, since m2 and m2_1 have the same overall accuracy, the choice of m2 as better model comes up after comparing Balanced accuracy and F1 score values. For the *random forests models*, the choice is clear and without ambiguity since m3_2 is also the model which gets among all the fitted models in our study, the highest overall accuracy value, 0.7833. The group of *SVM models* indicates that m4_2, the support vector machine model with vanilladot Linear kernel function, shows the best overall accuracy value as well as best F1-score and balanced accuracy values.

- **AUC, KS, GINI**

From the comparative study , we found that *random forests-tuning parameters* performing better than other methods with given set-up. This random forest model is constitued by the 5 top variables based on MeanDecrease Gini , which is an average mean of a variable's total decrease in node impurity, weighted by the proportion of samples reaching that node in each individual decision tree in the random forest. A higher mean decrease in GIni indicates a higher variable importance. This model shows a good overall accuracy, 0.7833 and the highest performance on AUC and Gini values , i.e 0.7813 and 0.5626, respectively.(Even if its KS value is the third after the one of m4_2 and m1 models). Also, we found that the *SVM-vanilladot* results to be the second best classifier after m3_2, since to equal overall accuracy value with Logistic regression model m1, it shows higher values of AUC, KS and GINI, i.e 0.7794, 0.4857 and 0.5588, respectively. Finally, comparing m1 and m6 metrics values, at first eye, we could choose logistic regression model as better model.

So, why we prefer Lasso than logistic? : The output of `coef(lasso_model,s=lambda_1se)` (see below) shows that 9 out of 12 variables that we had determined to be significant based on the basis of p-values (in m1 model) have non-zero coefficients. The coefficients of other variables have been set to zero by the algorithm. Lasso has reduced the complexity of the fitting function massively. That explained too the effect on the overall accuracy value of m6 , which is a less than what we got with m1. So, we get a similar out-of-sample accuracy (0,7567 m1 vs 0.7267 m6, i.e a difference of only 0.039%) as we did before, and we do so using a way simpler function(9 non-zero coefficients) than the orginal one(12 non zero coefficients, 15 non zero coefficients if we take into account all predictors in m1,significant or not). What this means is that the simpler function does at least as good a job fitting the signal in the data as the more complicated one. The bias-variance tradeoff tell us that the simpler function should be preferred because it is less likely to overfit the training data. Read more here

Thats why, at the end, after models m3_2 and m4_2, we choose as third best classifier the lasso regression model , m6.

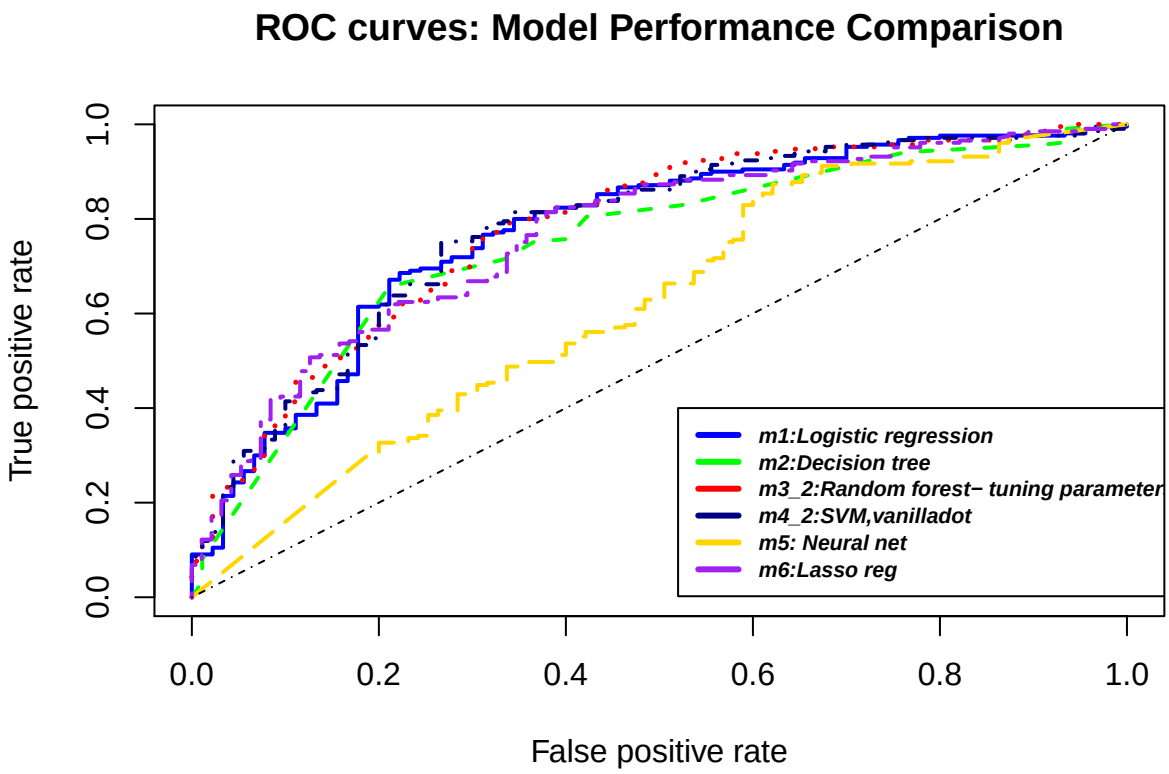
```
 "fitted lasso model - coefficients with lambda.1se"

17 x 1 sparse Matrix of class "dgCMatrix"
                            1
(Intercept)      -0.90543365
(Intercept)       .
check_acct        0.42029673
credit_hist       0.23474145
Duration         -0.02074873
savings_acct      0.08818443
Purpose           .
Age               .
Property         -0.04136937
Amount            .
present_emp       0.01521461
Housing           .
other_install     0.12194408
status_sex        0.02845865
ForeignWorker    -0.05414408
other_debt        .
installment_rate  .
```

Models	AUC	KS	Gini
m1:Logistic regression	0.7741	0.4635	0.5482
m2:Decision tree	0.7477	0.446	0.4954
m3_2: Random forest- tuning parameters	0.7813	0.4571	0.5626
m4_2:SVM,vanilladot	0.7794	0.4857	0.5588
m5: Neural net	0.6185	0.2521	0.237
m6:Lasso reg	0.7675	0.4357	0.535

Table 2: Comparison of Model Performances- best models

5.2.2 Model Performance Comparison - ROC curves

In paragraph 4.2, we already defined and explained what is the Receiver Operating Characteristic (ROC) curve. Here, we drew a model performance comparison based on the ROC curves of selected most accurate models: m1, m2, m3_2, m4_2, m6. We also added the ROC curve of the Neural network model (m5), the least accurate model. The "ROC curves: Model performance comparison" plot confirms our previous analysis.



6 Conclusions and suggestions

This German credit risk project has examined the potential best Machine learning algorithm to classify credit worthiness for the 1000 applicants of the **rchallenge** version of german credit data. Using the splitted training set and test set, we successively trained different classifiers algorithms (logistic regression, support vector machines, decision trees, Neural networks, Lasso regression) and predict each fitted model on test set. The model evaluation performance through the overall accuracy, F1-score and balanced accuracy values, and optimization through AUC, KS and GINI values showed that random forest with tuning parameters, support vector machine based on vanilla linear kernel function and lasso regression, are the three appropriate machine learning algorithms to classify at best the credit risk of 1000 applicants to loan.

Future work includes further investigations on Ensemble methods by following the work of Wang et al(2011). In fact, in the latter, a comparative assessment of three popular ensemble methods, i.e. Bagging, Boosting, and Stacking, based on four base learners, i.e., Logistic Regression Analysis (LRA), Decision Tree (DT), Artificial Neural Network (ANN) and Support Vector Machine (SVM) could possibly carried out an overall accuracy greater than 0.8. However, we should precise that these results comes up by combining three datasets : German and Australian credit datasets, which are from UCI machine learning repository, and China credit dataset, which is from Industrial and Commercial Bank of China.

Several future research directions also emerge . Firstly, large datasets for experiments and applications, particularly with more exploration of credit scoring data structures, should be collected to further valid the

conclusions of our study. Secondly, further analyses are encouraged to explore the reasons why the Neural Network model has the worst performance for all accuracy metrics.

References

- Ha, V. S., & Nguyen, H. N. (2016). Credit scoring with a feature selection approach based deep learning. In MATEC Web of Conferences (Vol. 54, p. 05004). EDP Sciences.
- Aggarwal, C. C. (Ed.). (2014). Data classification: algorithms and applications. CRC press.
- Wang, G., Hao, J., Ma, J., & Jiang, H. (2011). A comparative assessment of ensemble learning for credit scoring. Expert systems with applications, 38(1), 223-230.
- Therneau, T. M., & Atkinson, E. J. (2018). An Introduction to Recursive Partitioning Using the RPART Routines.
- Vapnik, V. (1998). The nature of statistical learning theory. Springer science & business media.
- Karatzoglou, A., Meyer, D., & Hornik, K. (2005). Support vector machines in R.
- Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. ACM transactions on intelligent systems and technology (TIST), 2(3), 27.
- Meyer, D., & Wien, F. T. (2015). Support vector machines. The Interface to libsvm in package e1071, 28.
- Cortes, C. & Vapnik, V. (1995). Support-vector network. Machine Learning, 20, 1–25.
- Ripley, B., Venables, W., & Ripley, M. B. (2016). Package 'nnet'. R package version, 7, 3-12.
- Soni, A. K., & Abdullahi, A. U. (2015). Using neural networks for credit scoring. Int. J. Sci. Technol. Manag, 4.
- Liu, Y. (2002). The evaluation of classification models for credit scoring. Institut für Wirtschaftsinformatik, Georg-August-Universitat Göttingen.
- Bock, H. H. (2001). Construction and Assessment of Classification Rules. David J. Hand, Wiley, Chichester, 1997. No. of pages: xii+ 214. Price:£ 34.95. ISBN 0-471-96583-9. Statistics in Medicine, 20(2), 326-327.
- Sivakumar,S.,2015,German credit Data Analysis,github page, http://srisai85.github.io/GermanCredit/German.html
- Liaw A, Wiener M (2002). "Classification and Regression by randomForest." R News, 2(3),18–22. URL http://CRAN.R-project.org/doc/Rnews/.
- Kursa, M. B., & Rudnicki, W. R. (2010). Feature selection with the Boruta package. J Stat Softw, 36(11), 1-13.
- Siddiqi, Naeem (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. SAS Institute, pp 79-83.
- Therneau, T. M., & Atkinson, E. J. (2018). An Introduction to Recursive Partitioning Using the RPART Routines.

- Hand, David J.; and Till, Robert J. (2001); A simple generalization of the area under the ROC curve for multiple class classification problems, Machine Learning, 45, 171–186.
- Hand, D. J., & Henley, W. E. (1997). Statistical classification methods in consumer credit scoring: a review. Journal of the Royal Statistical Society: Series A (Statistics in Society), 160(3), 523-541.
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1), 267-288.
- Massey Jr, F. J. (1951). The Kolmogorov-Smirnov test for goodness of fit. Journal of the American statistical Association, 46(253), 68-78.
- Günther, F., & Fritsch, S. (2010). neuralnet: Training of neural networks. The R journal, 2(1), 30-38.
- Beck, M. W. (2018). NeuralNetTools: Visualization and analysis tools for neural networks. Journal of statistical software, 85(11), 1.
- Soni, A. K., & Abdullahi, A. U. (2015). Using neural networks for credit scoring. Int. J. Sci. Technol. Manag, 4.