A Model to Identify Risky Loans

Emilia Iacob

March 8, 2023

Contents

Introduction

This is a report that describes the steps taken to develop a prediction model capable of identifying risky bank loans. This model uses a cleaner version of the german credit dataset donated to the UCI Machine Learning Repository by Hans Hofmann from the University of Hamburg. The original dataset can be found on the [UCI Machine Learning Repository] (https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data). The cleaner version of the german credit dataset used in this report is available thanks to Brett Lantz and his book "Machine Learning with R - Expert techniques for predictive modeling", 3rd edition, Packt Publishing - see the References section below.

The dataset contains information about 1000 loans, along with demographics about their corresponding loan applicants, the size, purpose and period of the loan and if those loans eventually went into default or no.

The method that we'll use to judge how well our algorithm is able to predict default loans will be the overall accuracy and sensitivity (i.e. the percentage of defaulted loans that have been correctly predicted as default). These measures will be applied on a test set - a dataset the we have not used when building our algorithm but for which we already know the actual default status of the loans.

The approach used to build our prediction algorithm takes into account a few steps: - first we split our credit data into a train dataset (90% of the data) and a final holdout test set (remaining 10% of the data).
- we further split our train dataset into training and testing datasets to be able to train and test our algorithms. - next we use 2 implementations of the decision tree algorithms: C5.0 and Random Forests. Unlike other classification machine learning algorithms which are like black boxes (i.e. hard to understand the way classification decisions have been made) the advantage of using decision trees is that the results are easy to read in plain language. Initially we will use the default parameters of each algorithm and later we will use cross-validation to tune and pick the parameters that give us the highest accuracy and sensitivity.
- in the end we compare the algorithms and we choose the best one to test on the final holdout test set.

The objective will be to build a predictive model that would give us over 75% accuracy and the highest possible sensitivity meaning that we would be able to correctly predict the highest number of loans that have defaulted.

Analysis

Step 1: Collecting and preparing the data

On [*Kaggle*] (https://www.kaggle.com/datasets/uciml/german-credit/download?datasetVersionNumber=1) there is a clean version of the german dataset but it only has 9 attributes. We are going to use instead the clean version that has 16 attributes that has been made available by Brett Lantz in his book "Machine Learning with R - Expert techniques for predictive modeling", 3rd edition, Packt Publishing - see the *References* section below.

Let's first call the packages that we are going to use in this analysis:

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(C50)) install.packages("C50", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(GGally)) install.packages("GGally", repos = "http://cran.us.r-project.org")
if(!require(gmodels)) install.packages("gmodels", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(C50)
library(randomForest)
library(data.table)
library(dplyr)
library (knitr)
library (GGally)
library(gmodels)
```

Now we can start downloading the file like this:

```r
dl <- "credit.csv"
if(!file.exists(dl))
  download.file(https://raw.githubusercontent.com/emilia-iacob/LoanRisk/main/credit.csv", dl)
```

Let's create the *credit* object:

```r
credit <- read.csv("credit.csv", stringsAsFactors = TRUE)
```

If we want to save the credit object for using it later we can do it like this:

```r
save(credit, file = "credit.rda")
```

To load the file when opening a new session:

```r
load("credit.rda")
```

Let's now create the train dataset (*credit_main*) on which we will train and test our algorithms and will be 90% of the credit data set and let's also create the final holdout test set (*credit_final_holdout_set*) which will be the remaining 10% of the credit data set and will be used to evaluate the performance of our final algorithm:

```r
set.seed (123, sample.kind = "Rounding")
test_index <- createDataPartition(y=credit$default, times = 1, p = 0.1, list = FALSE)
credit_main <- credit [-test_index,]
credit_final_holdout_test <- credit [test_index,]
```

Now that we have created the train dataset (aka *credit_main*) we'll split it further into a train and test dataset with 90% and 10% of the *credit_main* dataset respectively:

```r
set.seed(123, sample.kind = "Rounding")
main_test_index <- createDataPartition( y= credit_main$default, times = 1, p = 0.1, list = FALSE)
credit_main_train <- credit_main [-main_test_index,]
credit_main_test <- credit_main [main_test_index,]
```

To save the resulting files (*credit_main*, *credit_main_test*, *credit_main_train* and *final_holdout_test*) for future use we'll do the following:

```r
save(credit_main, file = "credit_main.rda")
save(credit_final_holdout_test, file = "credit_final_holdout_test.rda")
save(credit_main_train, file = "credit_main_train.rda")
save(credit_main_test, file = "credit_main_test.rda")
```

To quickly load the saved *edx* and *final_holdout_test* files and go to the next steps of this analysis we can use the load function:

```r
load("credit_main.rda")
load("credit_main_train.rda")
load("credit_main_test.rda")
load("credit_final_holdout_test.rda")
```

Step 2: Exploring the credit dataset

Let's first do some exploratory data analysis on the *credit* dataset to better understand the challenge that we have ahead.

First, let's see how many observations and variables we have in our *credit* dataset:

```
dim(credit)
```

```
## [1] 1000   17
```

Next let's see the type of data we have in the *credit* dataset:

```
str(credit)
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4 1 1 4 4 3 4 3 ...
##  $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credit_history      : Factor w/ 5 levels "critical","good",..: 1 2 1 2 4 2 2 2 2 1 ...
##  $ purpose             : Factor w/ 6 levels "business","car",..: 5 5 4 5 2 4 5 2 5 2 ...
##  $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ savings_balance     : Factor w/ 5 levels "< 100 DM","> 1000 DM",..: 5 1 1 1 1 5 4 1 2 1 ...
##  $ employment_duration : Factor w/ 5 levels "< 1 year","> 7 years",..: 2 3 4 4 3 3 2 3 4 5 ...
##  $ percent_of_income   : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ years_at_residence  : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ other_credit        : Factor w/ 3 levels "bank","none",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ housing             : Factor w/ 3 levels "other","own",..: 2 2 2 1 1 1 2 3 2 2 ...
##  $ existing_loans_count: int  2 1 1 1 2 1 1 1 1 2 ...
##  $ job                 : Factor w/ 4 levels "management","skilled",..: 2 2 4 2 2 4 2 1 4 1 ...
##  $ dependents          : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ phone               : Factor w/ 2 levels "no","yes": 2 1 1 1 1 2 1 2 1 1 ...
##  $ default             : Factor w/ 2 levels "no","yes": 1 2 1 1 2 1 1 1 1 2 ...
```

We see that *checking_balance* and *savings_balance* are all factors.

Let's now see how many of the loans in the *credit* dataset have defaulted:

```
table(credit$default)
```

```
## 
##  no yes 
## 700 300
```

So 30% of the loans that we have in our dataset have defaulted.

Let's now see the distribution of the checking balance in percentages:

```
prop.table(table(credit$checking_balance))*100
```

```
## 
##     < 0 DM   > 200 DM 1 - 200 DM    unknown 
##       27.4        6.3       26.9       39.4
```

We can see that only 6% of the applicants for a loan had a checking balance over 200 DM which was the German currency at the time, before Germany adhered to the European Union. However, in almost 40% of the cases we don't have any information about *checking_balance*.

Let's now explore the distribution of the same checking balance by default status:

```
credit %>% ggplot(aes(x=checking_balance, fill = default)) +
  geom_bar(position = "dodge") +
  ggtitle ("Checking balance distribution by loan default status")
```

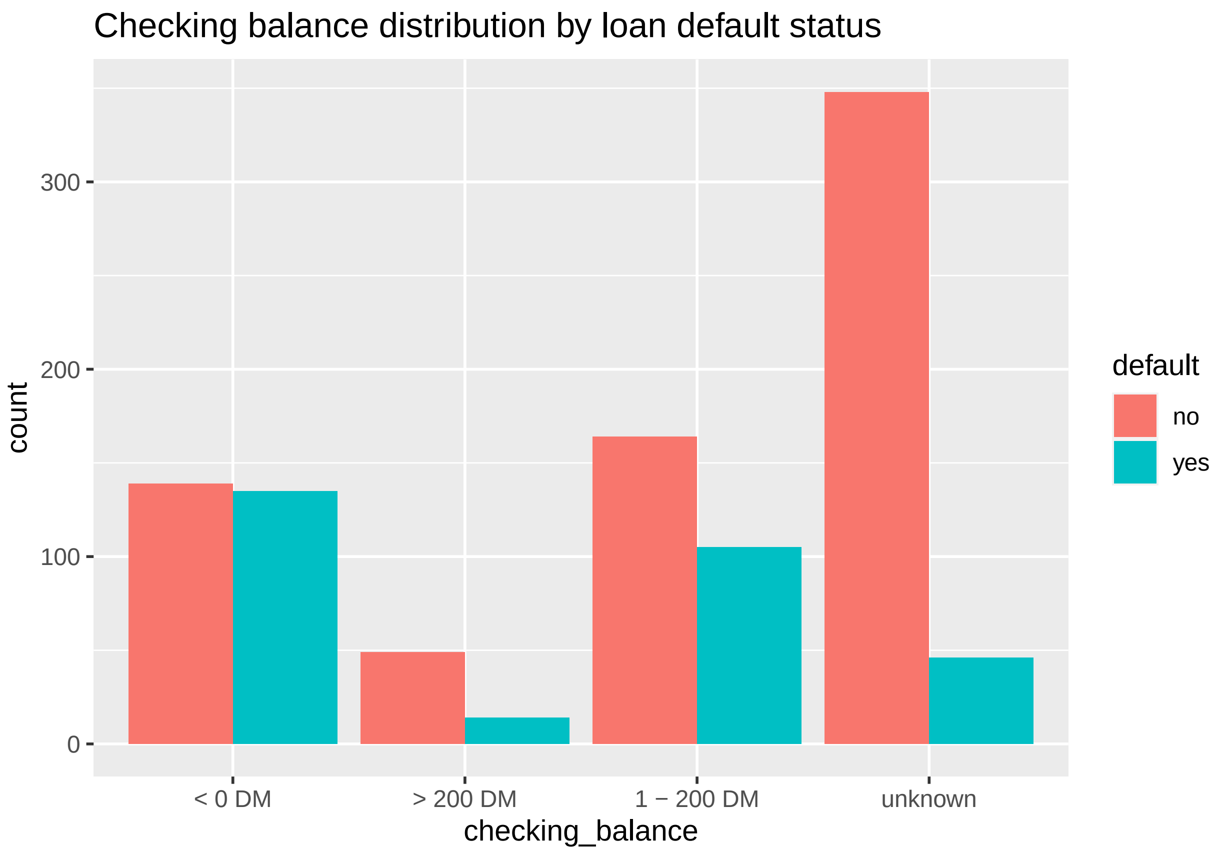


Figure 1: Distribution of checking balance by loan default status

As we are already expecting, we notice that the default status is the lowest for those applicants with the highest checking balance (over 200 DM). However, it is surprising to see that even among the applicants for which we don't have any information about their checking balance the default status is low.

Let's see a similar view for the savings balance:

```
credit %>% ggplot(aes(x=savings_balance, fill = default)) +
  geom_bar(position = "dodge")+
  ggtitle("Savings balance distribution by loan default status")
```

Here again, we see that those applicants with a savings balance over 500 DM have the lowest default rate compared to the applicants in other savings balance categories.

Let's now look at the loan distribution by purpose:

```
credit %>% ggplot(aes(x=purpose, fill = default)) +
  geom_bar()+
  facet_grid(rows = vars(default)) +
  ggtitle("Distribution by loan purpose and default status")
```

Let's see the distribution by credit history:

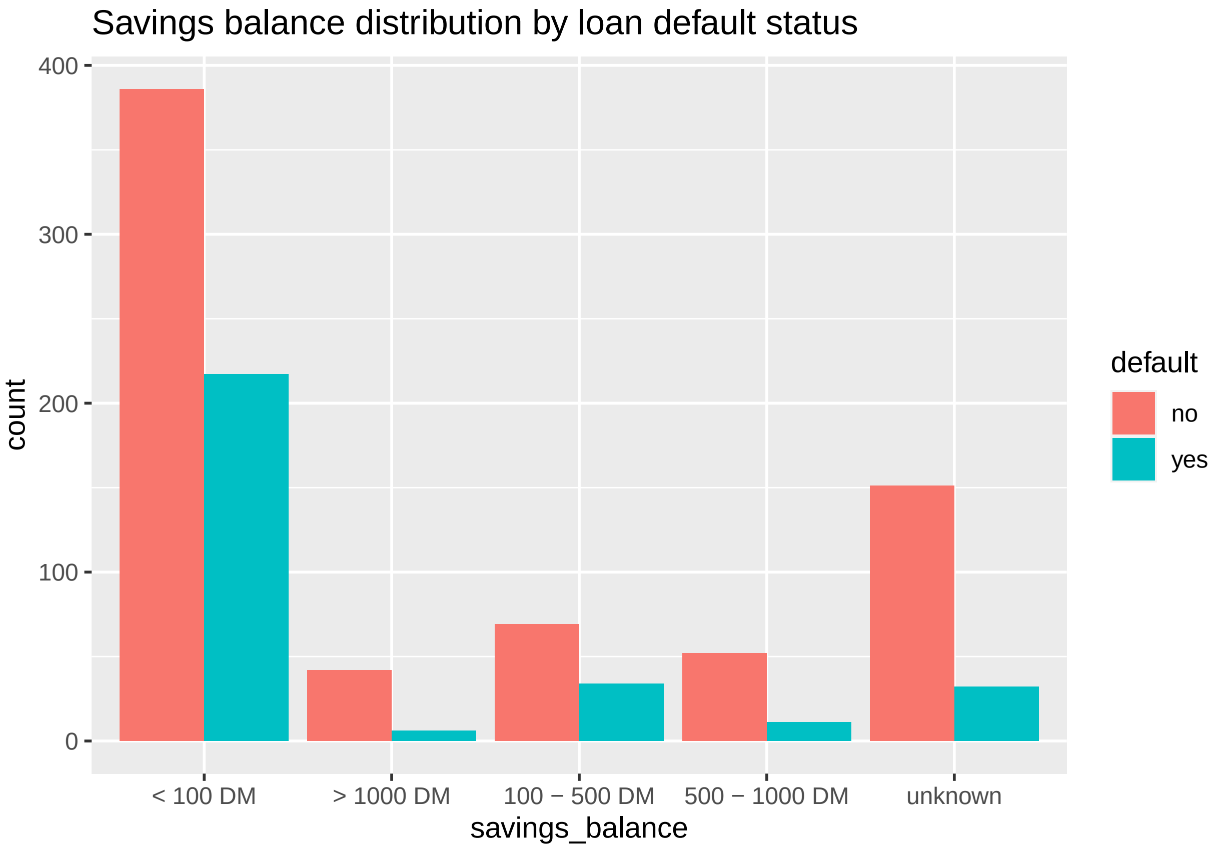


Figure 2: Distribution of savings balance by loan default status

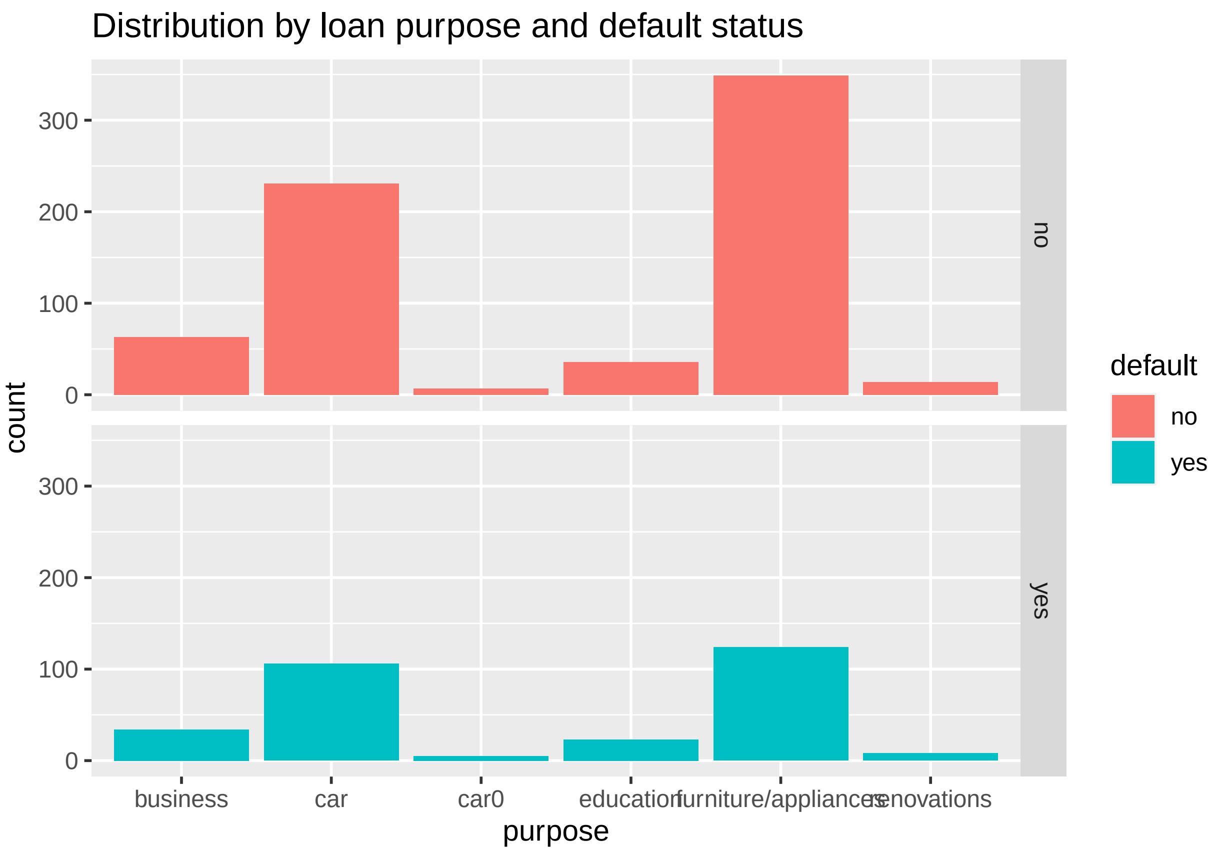


Figure 3: Loan distribution by purpose and default status

```
credit %>% ggplot(aes(x=credit_history, fill = default)) +
  geom_bar(position = "dodge") +
  ggtitle("Loan Distribution by credit history and default status")
```

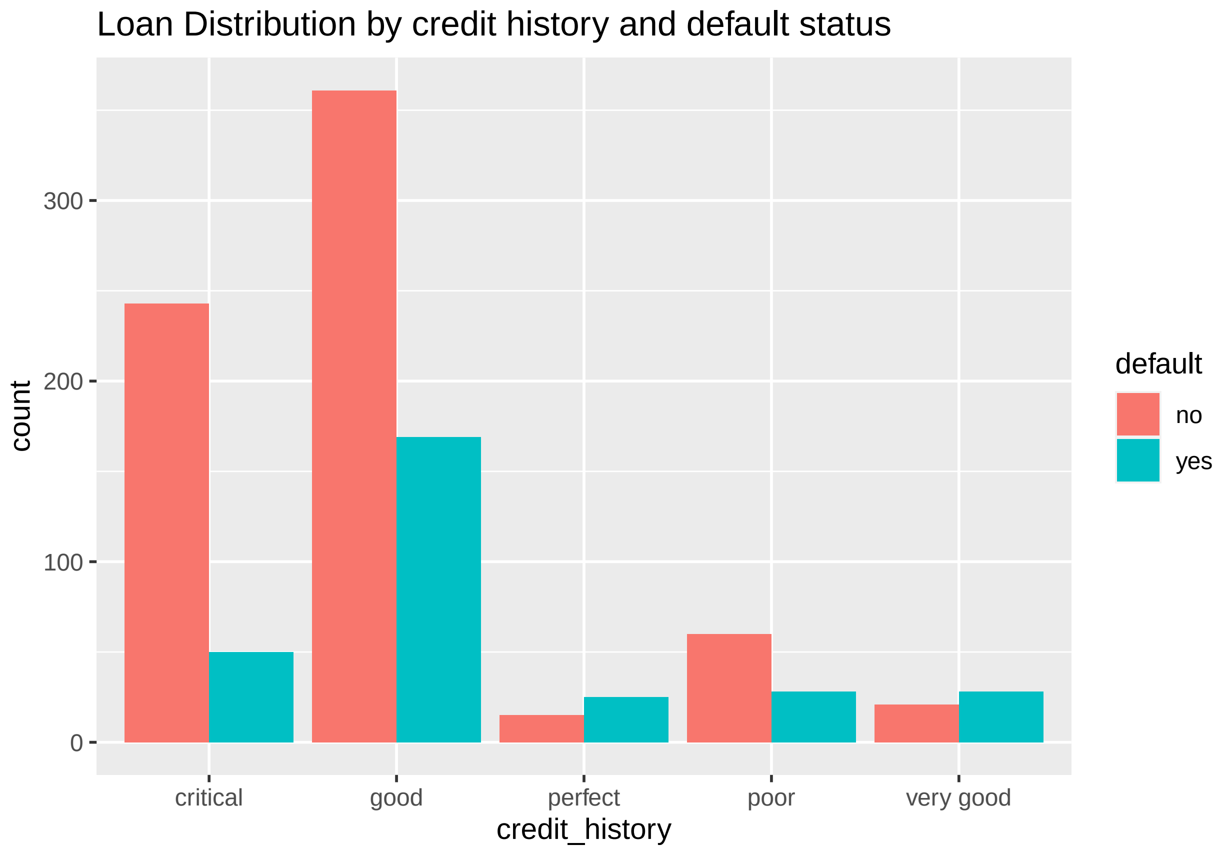


Figure 4: Loan distribution by credit history and default status

Here it's interesting to see that those applicants having a very good or perfect credit history actually default more than the applicants in any other category. We also notice that the number of loans given to applicants with good and critical credit history is disproportionally higher than the loans given to applicants in other categories. This may also be because people with good credit history tend to have a better money discipline and may not apply to loans in the first place.

Let's now look at the same distribution of credit history by default status with the CrossTable function from the gmodels package. The CrossTable function also offers the calculation for Pearson's chi-squared test for independence between 2 variables. This test tells us how likely it is that the difference in cell counts in the table is due to chance alone. The lower the chi-squared values the higher the evidence that there is an association between the 2 variables.

```
prop.table(table(credit$credit_history, credit$default))
```

```
##
##                no   yes
##   critical  0.243 0.050
##   good      0.361 0.169
##   perfect   0.015 0.025
##   poor      0.060 0.028
##   very good 0.021 0.028
```

```r
CrossTable(x=credit$default, y= credit$credit_history, chisq = TRUE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1000
##
##
##                | credit$credit_history
## credit$default |  critical |      good |   perfect |      poor | very good | Row Total |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|
##             no |       243 |       361 |        15 |        60 |        21 |       700 |
##                |     7.003 |     0.270 |     6.036 |     0.042 |     5.157 |           |
##                |     0.347 |     0.516 |     0.021 |     0.086 |     0.030 |     0.700 |
##                |     0.829 |     0.681 |     0.375 |     0.682 |     0.429 |           |
##                |     0.243 |     0.361 |     0.015 |     0.060 |     0.021 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|
##            yes |        50 |       169 |        25 |        28 |        28 |       300 |
##                |    16.341 |     0.629 |    14.083 |     0.097 |    12.033 |           |
##                |     0.167 |     0.563 |     0.083 |     0.093 |     0.093 |     0.300 |
##                |     0.171 |     0.319 |     0.625 |     0.318 |     0.571 |           |
##                |     0.050 |     0.169 |     0.025 |     0.028 |     0.028 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|
##   Column Total |       293 |       530 |        40 |        88 |        49 |      1000 |
##                |     0.293 |     0.530 |     0.040 |     0.088 |     0.049 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  61.6914     d.f. =  4     p =  1.279187e-12
##
##
##
```

Here we see that the p value is less than 0.05 which means that we can reject the null hypothesis meaning that it's very likely that the variation in cell count for the 2 variables might not be due to chance.

Let's now look at the numeric variables in the *credit* dataset and create a correlogram with ggpairs:

```r
credit_1 <- credit %>% select(default, months_loan_duration, amount,
                              percent_of_income, age, existing_loans_count)
ggpairs(credit_1, ggplot2::aes(colour= default))
```

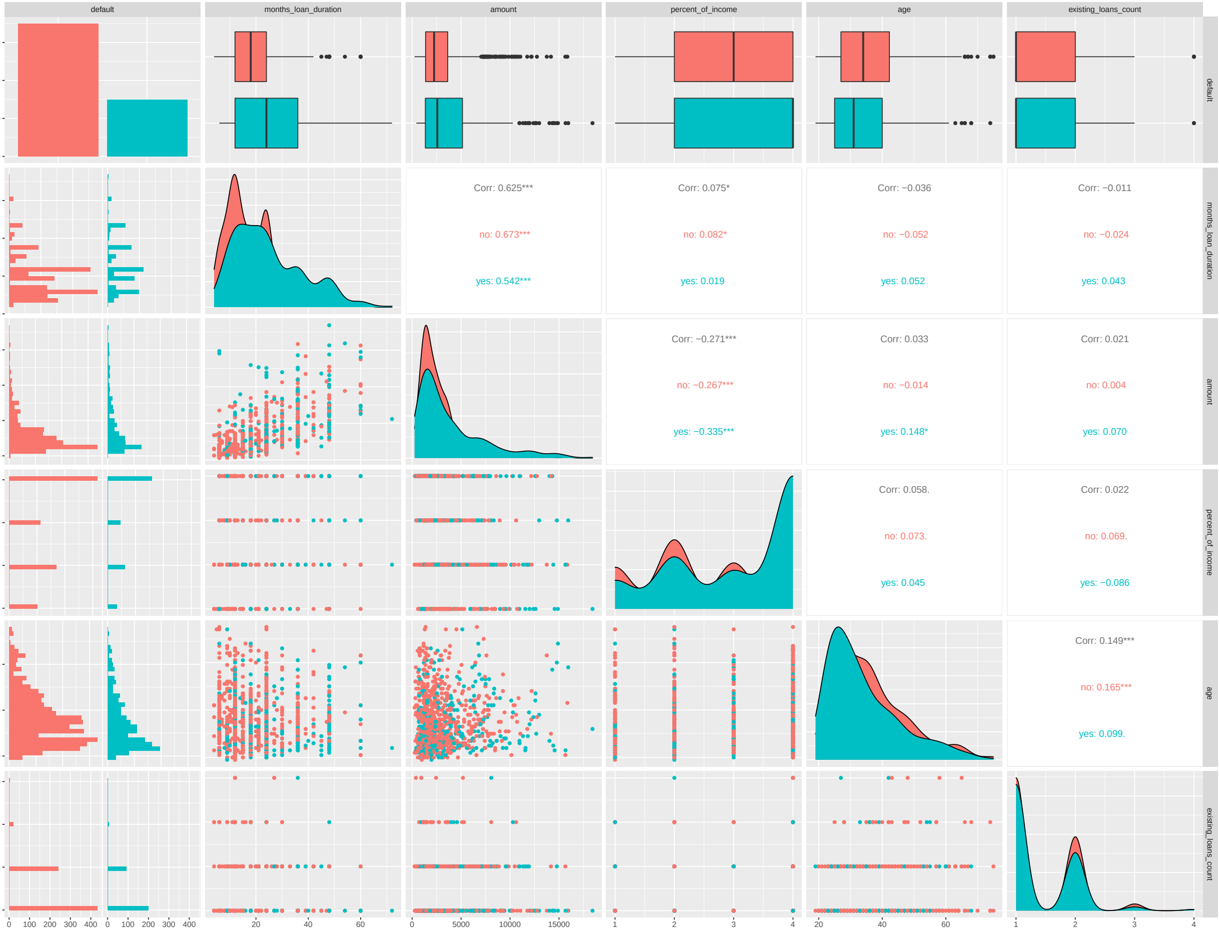


Figure 5: Correlogram for numerical variables

Here we see that the only significant correlation is between the duration of a loan and its amount which is expected since usually the higher the loan amount the longer it may take to repay it.

Let's see the distribution of the amount of loan based on default status:

```
credit %>% ggplot(aes(x= default, y= amount, fill = default)) +
  geom_boxplot() +
  ggtitle("Distribution of loan amount based on default status")
```

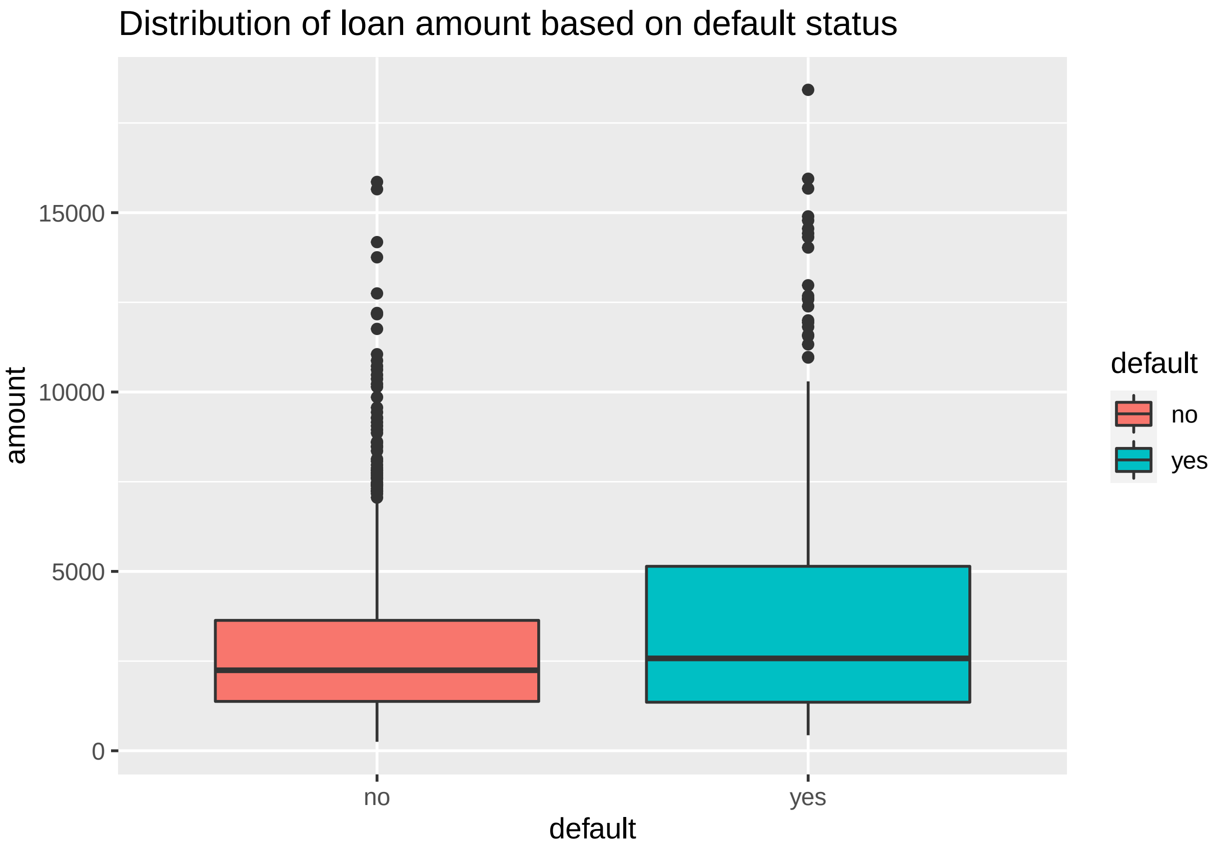


Figure 6: Loan amount distribution by default status

We can see that there is no significant difference between the average amount of a loan that defaulted versus a loan that has been paid out:

- loans defaulted:

```
credit %>% filter(default == "yes") %>% select (amount) %>% summary(credit$amount)
```

```
##       amount
##   Min.   :  433
##   1st Qu.: 1352
##   Median : 2574
##   Mean   : 3938
##   3rd Qu.: 5142
##   Max.   :18424
```

- loans successfully paid out:

```
credit %>% filter(default == "no") %>% select (amount) %>% summary(credit$amount)
```

```
##      amount
##  Min.   :  250
##  1st Qu.: 1376
##  Median : 2244
##  Mean   : 2985
##  3rd Qu.: 3635
##  Max.   :15857
```

Let's see the distribution of the number of months for a loan based on default status:

```
credit %>% ggplot(aes(x= default, y=months_loan_duration, fill = default)) +
  geom_boxplot() +
  ggtitle("Loan duration by default status")
```

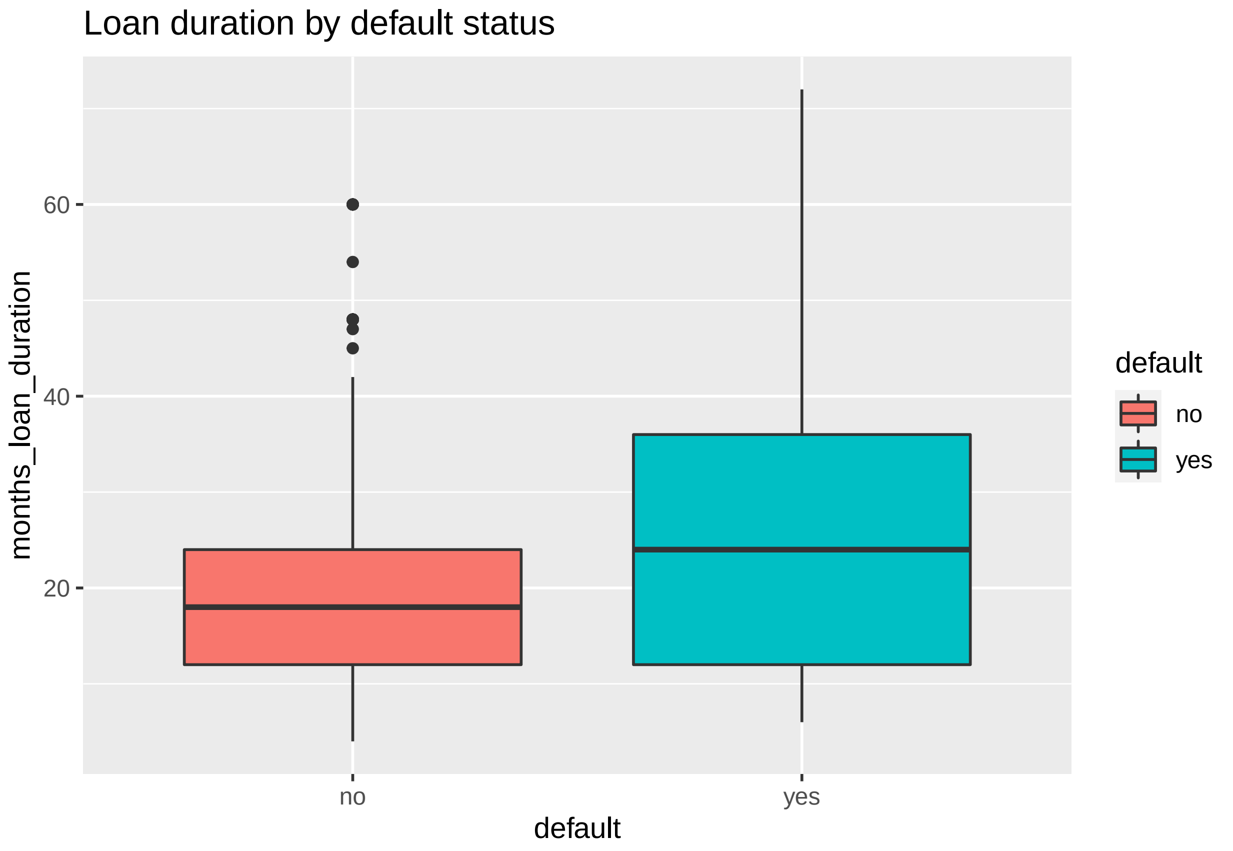


Figure 7: Loan duration by default status

We see that the loans that defaulted have on average a higher duration than the loans that have been paid out.

```
credit %>% ggplot(aes(x=age, fill = default)) +
  geom_bar(position = "dodge") +
  ggtitle ("Age distribution by the loan default status")
```

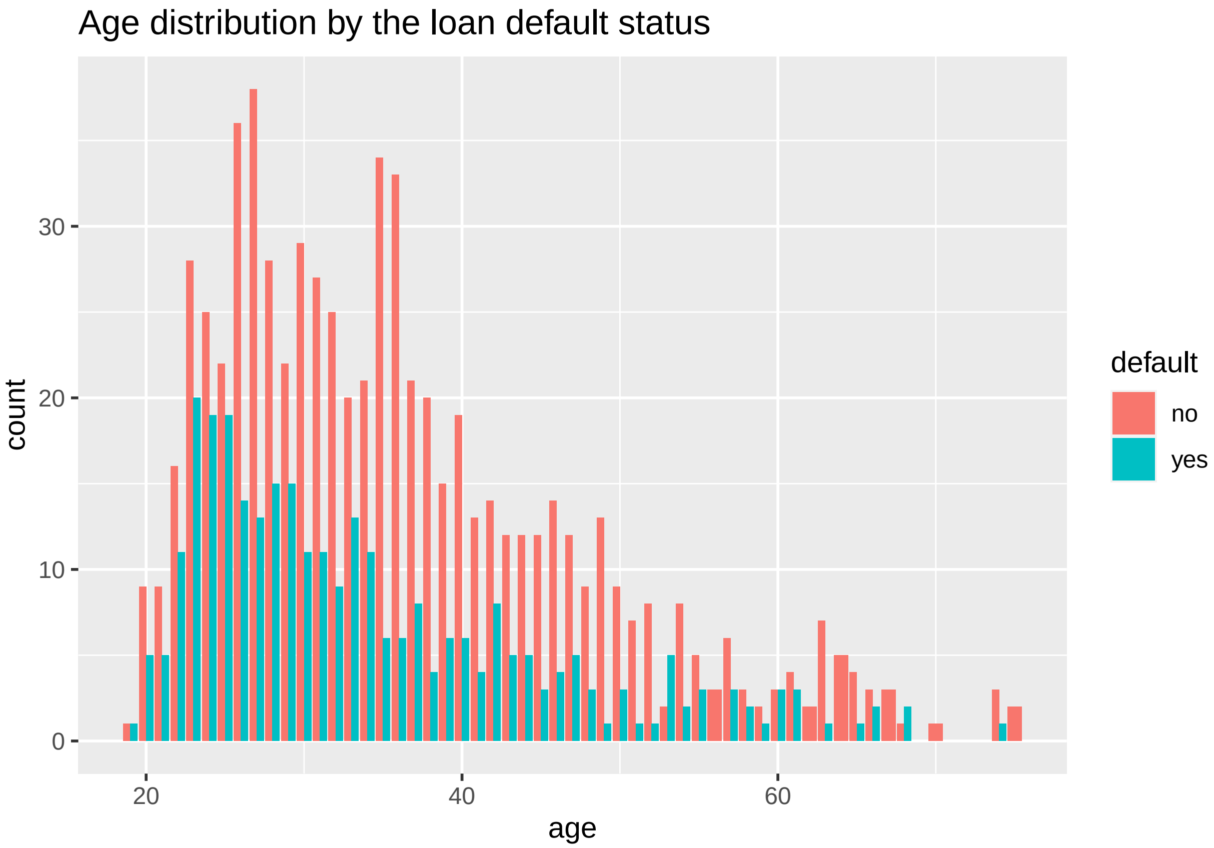


Figure 8: Applicants age distribution by loan default status

Step 3: Training and evaluating different models

First Model: C5.0 Decision Trees Algorithm (default parameters)

Let's start by training a C5.0 decision tree model with the default parameters from the *caret* package:

```
set.seed(123, sample.kind = "Rounding")
train_dt <- train(default ~ ., data = credit_main_train, method = "C5.0")
```

and let's see a summary of the results:

```
train_dt
```

```
## C5.0
##
## 810 samples
##  16 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 810, 810, 810, 810, 810, 810, ...
## Resampling results across tuning parameters:
##
##   model  winnow  trials  Accuracy   Kappa
##   rules  FALSE    1      0.6876429  0.2562157
##   rules  FALSE   10      0.7091948  0.2992349
##   rules  FALSE   20      0.7132118  0.3026864
##   rules   TRUE    1      0.6848251  0.2506519
##   rules   TRUE   10      0.7124136  0.3039926
##   rules   TRUE   20      0.7192568  0.3179016
##   tree   FALSE    1      0.6764555  0.2192561
##   tree   FALSE   10      0.7180396  0.2800916
##   tree   FALSE   20      0.7255147  0.2983111
##   tree    TRUE    1      0.6823078  0.2348795
##   tree    TRUE   10      0.7103501  0.2680768
##   tree    TRUE   20      0.7219369  0.2944902
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were trials = 20, model = tree and winnow
##  = FALSE.
```

Let's also plot these results:

```
ggplot(train_dt) +
  ggtitle("C5.0 Decision Trees Algorithm")
```

To see the resulting tree let's access the finalModel:

```
train_dt$finalModel
```

```
##
## Call:
## (function (x, y, trials = 1, rules = FALSE, weights = NULL, control
##  2, fuzzyThreshold = FALSE, sample = 0, earlyStopping = TRUE, label
##  = "outcome", seed = 1210L))
##
## Classification Tree
```

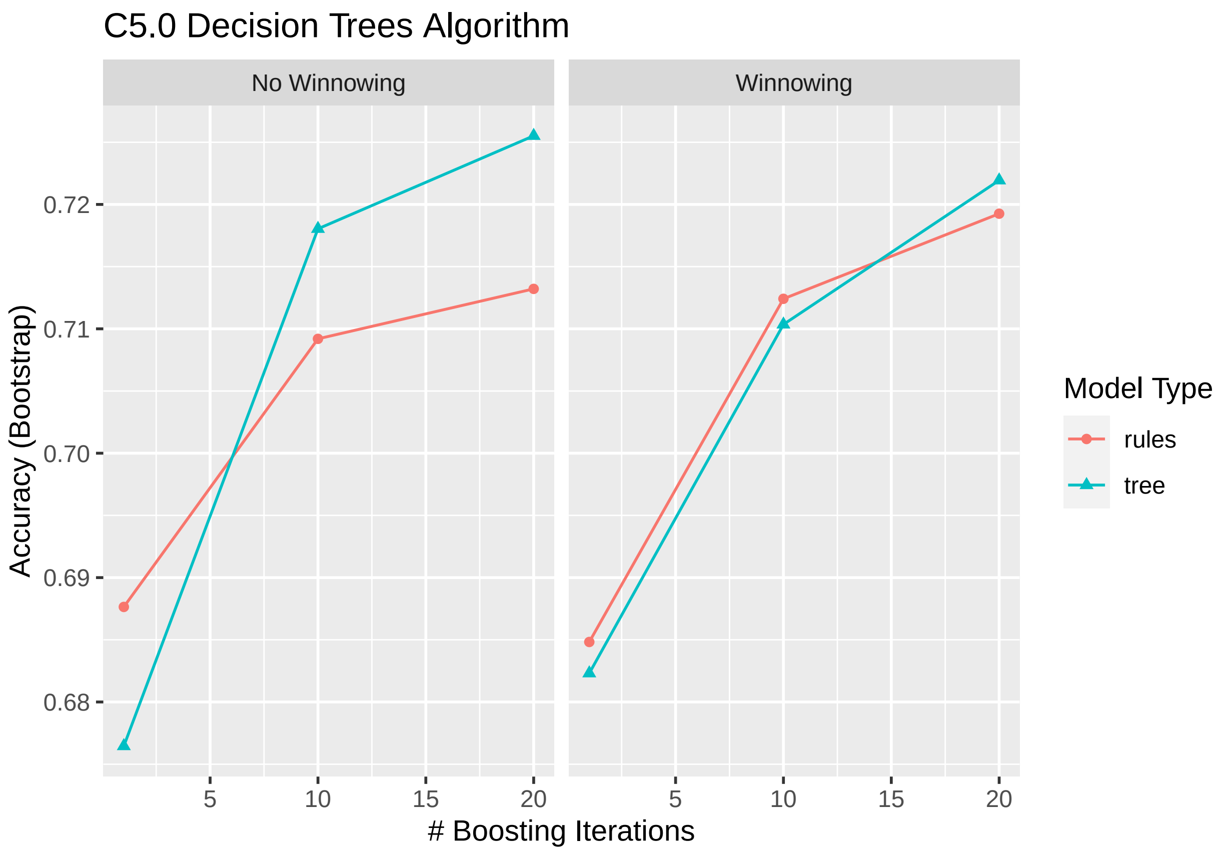


Figure 9: C5.0 Decision Trees Training Algorithm results

```
## Number of samples: 810
## Number of predictors: 35
##
## Number of boosting iterations: 20
## Average tree size: 57.5
##
## Non-standard options: attempt to group attributes
```

We can see that the average tree size is 66 and the number of boosting iterations used was 20. We also see that the number of predictors used are 35 now. Let's list these predictors:

```
train_dt$finalModel$predictors
```

```
##  [1] "checking_balance> 200 DM"       "checking_balance1 - 200 DM"
##  [3] "checking_balanceunknown"        "months_loan_duration"
##  [5] "credit_historygood"             "credit_historyperfect"
##  [7] "credit_historypoor"             "credit_historyvery good"
##  [9] "purposecar"                     "purposecar0"
## [11] "purposeeducation"               "purposefurniture/appliances"
## [13] "purposerenovations"             "amount"
## [15] "savings_balance> 1000 DM"       "savings_balance100 - 500 DM"
## [17] "savings_balance500 - 1000 DM"   "savings_balanceunknown"
## [19] "employment_duration> 7 years"   "employment_duration1 - 4 years"
## [21] "employment_duration4 - 7 years" "employment_durationunemployed"
## [23] "percent_of_income"              "years_at_residence"
## [25] "age"                            "other_creditnone"
## [27] "other_creditstore"              "housingown"
## [29] "housingrent"                    "existing_loans_count"
## [31] "jobskilled"                     "jobunemployed"
## [33] "jobunskilled"                   "dependents"
## [35] "phoneyes"
```

And let's see the importance of each of these predictors:

```
varImp(train_dt$finalModel)
```

```
##                                 Overall
## checking_balanceunknown          100.00
## months_loan_duration             100.00
## credit_historyperfect            100.00
## credit_historypoor               100.00
## amount                           100.00
## savings_balance>1000DM           100.00
## savings_balanceunknown           100.00
## credit_historyverygood            99.88
## checking_balance>200DM            99.75
## purposeeducation                  99.75
## employment_duration4-7years       99.75
## dependents                        99.14
## years_at_residence                98.89
## employment_durationunemployed     98.52
## other_creditstore                 98.15
## housingrent                       98.15
## other_creditnone                  97.90
## percent_of_income                 97.78
## jobunskilled                      96.91
## credit_historygood               95.43
## age                              94.44
## savings_balance100-500DM         94.20
## existing_loans_count             93.09
## employment_duration>7years       90.49
## employment_duration1-4years      88.77
## jobskilled                       87.78
## phoneyes                         82.47
## purposecar                       77.90
## savings_balance500-1000DM        74.57
## checking_balance1-200DM          73.58
## purposefurniture/appliances      63.33
## housingown                       60.12
## purposerenovations               50.74
## jobunemployed                    22.96
## purposecar0                      15.19
## checking_balance> 200 DM          0.00
## checking_balance1 - 200 DM        0.00
## credit_historyvery good           0.00
## savings_balance> 1000 DM          0.00
## savings_balance100 - 500 DM       0.00
## savings_balance500 - 1000 DM      0.00
## employment_duration> 7 years      0.00
## employment_duration1 - 4 years    0.00
## employment_duration4 - 7 years    0.00
```

We can see that the most often used predictors are *checking_balance unknown*, *savings_balance unknown*, *credit_history perfect*, *amount* and *months_loan_duration* and *purpose education*.

Let's now predict and evaluate our model on the test dataset:

```
predict_dt <- predict(train_dt, credit_main_test, type = "raw")

accuracy_dt <- confusionMatrix(predict_dt,
                               credit_main_test$default,
                               positive = "yes")$overall["Accuracy"]

sensitivity_dt <- sensitivity(predict_dt,
                              credit_main_test$default,
                              positive = "yes")
```

Let's create a table with the accuracy of each of the models that we build:

```
risk_models <- data.frame(Model = "C5.0 Decision Trees (default)",
                          Accuracy = accuracy_dt,
                          Sensitivity = sensitivity_dt)
risk_models %>% kable()
```

	Model	Accuracy	Sensitivity
Accuracy	C5.0 Decision Trees (default)	0.7111111	0.4814815

We see that our Decision Trees C5.0 model has a sensitivity rate of only 48% which means that only 48% of the default loans have been predicted correctly. Our model right now still cannot be deployed in real life as this would mean that we can only correctly predict 48% of the default loans which would be very costly for the bank. We need to do better than this.

Second Model: C5.0 Decision Trees Algorithm (tuned parameters)

Let's first see the parameters that can be tuned for the C5.0 model:

```
modelLookup("C5.0")
```

```
##   model parameter                 label forReg forClass probModel
## 1  C5.0    trials # Boosting Iterations  FALSE     TRUE      TRUE
## 2  C5.0     model            Model Type  FALSE     TRUE      TRUE
## 3  C5.0    winnow                Winnow  FALSE     TRUE      TRUE
```

We already saw that the best model with the default parameters of the C5.0 model had 20 trials, no winnowing and the model type was tree.

Let's create a tuning grid to optimize the *model*, *trials* and *winnow* parameters available for the C5.0 decision tree algorithm:

```
tune_grid <- expand.grid(model = "tree",
                         trials = seq(1,30, 2),
                         winnow = FALSE)
tune_grid
```

```
##    model trials winnow
## 1   tree      1  FALSE
## 2   tree      3  FALSE
## 3   tree      5  FALSE
## 4   tree      7  FALSE
## 5   tree      9  FALSE
## 6   tree     11  FALSE
## 7   tree     13  FALSE
## 8   tree     15  FALSE
## 9   tree     17  FALSE
## 10  tree     19  FALSE
## 11  tree     21  FALSE
## 12  tree     23  FALSE
## 13  tree     25  FALSE
## 14  tree     27  FALSE
## 15  tree     29  FALSE
```

Also, the *trainControl()* function in the caret package controls the parameters of the train() function.

Let's tune our model by using the *trainControl()* function to create a control object (named *control*) that uses 10 fold cross-validation

```
control <- trainControl(method = "cv", number = 10)
```

Let's now pass these tuned parameters to the train function again:

```
set.seed(123, sample.kind = "Rounding")
train_dt_tuned <- train(default ~., credit_main_train,
                        method = "C5.0",
                        trControl = control,
                        tuneGrid = tune_grid)

train_dt_tuned
```

```
## C5.0
##
## 810 samples
##  16 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 728, 728, 728, 730, 729, 729, ...
## Resampling results across tuning parameters:
##
##   trials  Accuracy   Kappa
##    1      0.6938603  0.2209424
##    3      0.7023649  0.2686908
##    5      0.7048645  0.2653649
##    7      0.6887681  0.2196704
##    9      0.6949413  0.2303325
##   11      0.7035693  0.2467139
##   13      0.7122727  0.2691844
##   15      0.7060994  0.2501415
##   17      0.7159617  0.2759569
##   19      0.7122433  0.2678131
##   21      0.7085535  0.2551058
##   23      0.7110230  0.2653686
##   25      0.7123340  0.2647437
##   27      0.7147576  0.2641726
##   29      0.7122738  0.2579932
##
## Tuning parameter 'model' was held constant at a value of tree
## Tuning
##  parameter 'winnow' was held constant at a value of FALSE
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were trials = 17, model = tree and winnow
##  = FALSE.
```

Let's now access the final model chosen and the best tune:

```
train_dt_tuned$finalModel
```

```
##
## Call:
## (function (x, y, trials = 1, rules = FALSE, weights = NULL, control
##  2, fuzzyThreshold = FALSE, sample = 0, earlyStopping = TRUE, label
##  = "outcome", seed = 664L))
##
## Classification Tree
## Number of samples: 810
## Number of predictors: 35
##
## Number of boosting iterations: 17
## Average tree size: 55.8
##
## Non-standard options: attempt to group attributes
```

```
train_dt_tuned$bestTune
```

```
##   trials model winnow
## 9     17  tree  FALSE
```

We can see that the bestTune for this model has now 17 trials compared to our default model which had 20 trials.

Also, because we are now using 10 fold cross-validation instead of the 25 bootstrapped samples, our sample size has been reduced to 728 compared to 810 in the default model. The size of the tree also dropped from 57 to 55 decisions deep.

Let's see how our new model performs :

```
predict_dt_tuned <- predict(train_dt_tuned, credit_main_test, type = "raw")
```

Let's now calculate accuracy and sensitivity for the new model:

```
accuracy_dt_tuned <- confusionMatrix(predict_dt_tuned,
                                     credit_main_test$default,
                                     positive = "yes")$overall["Accuracy"]
sensitivity_dt_tuned <- sensitivity(predict_dt_tuned,
                                    credit_main_test$default,
                                    positive = "yes")
```

and let's add them to our *risk_models* table:

```
risk_models <- rbind(risk_models, list("C5.0 Decision Trees (tuned)",
                                       accuracy_dt_tuned,
                                       sensitivity_dt_tuned))
risk_models %>% kable()
```

	Model	Accuracy	Sensitivity
Accuracy	C5.0 Decision Trees (default)	0.7111111	0.4814815
2	C5.0 Decision Trees (tuned)	0.7222222	0.5185185

We see that our tuned model has managed to slightly surpass the accuracy and sensitivity of the default model. However, we still need to do better because only 51% of the defaulted loans are currently predicted correctly.

Third Model: Rpart Regression Trees Algorithm (tuned parameters)

If we type:

```
modelLookup("rpart")
```

```
##   model parameter                label forReg forClass probModel
## 1 rpart        cp Complexity Parameter   TRUE     TRUE      TRUE
```

we see that the only tuning parameter available is cp (complexity paramenter). Let's use cross-validation to find the best cp:

```
set.seed(123, sample.kind = "Rounding")
train_rpart <- train(default ~., credit_main_train, method = "rpart",
                     tuneGrid = data.frame(cp = seq(0, 0.5, len = 25)))
```

If we plot the trained model we notice that the cp that gives the highest accuracy is 0.02

```
ggplot(train_rpart) +
  ggtitle("Rpart regression trees algorithm - tuned")
```

To see the complexity parameter that maximizes accuracy:

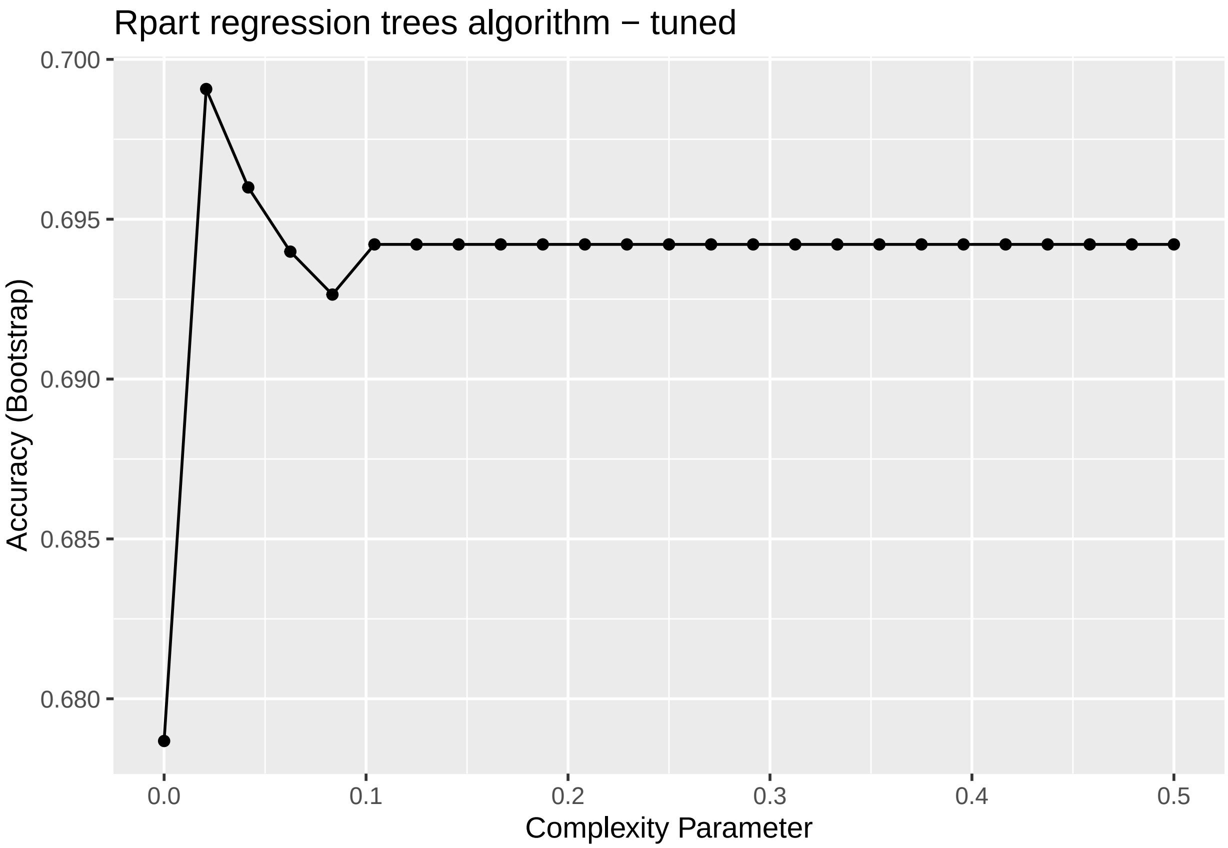


Figure 10: Rpart regression trees algorithm - tuned

```
train_rpart$bestTune
```

```
##           cp
## 2 0.02083333
```

Let’s now see the decisions in the resulting tree:

```
train_rpart$finalModel
```

```
## n= 810
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 810 243 no (0.7000000 0.3000000)
##    2) checking_balanceunknown>=0.5 316  38 no (0.8797468 0.1202532) *
##    3) checking_balanceunknown< 0.5 494 205 no (0.5850202 0.4149798)
##      6) months_loan_duration< 22.5 281  92 no (0.6725979 0.3274021) *
##      7) months_loan_duration>=22.5 213 100 yes (0.4694836 0.5305164)
##       14) checking_balance> 200 DM>=0.5 17   3 no (0.8235294 0.1764706) *
##       15) checking_balance> 200 DM< 0.5 196  86 yes (0.4387755 0.5612245)
##         30) savings_balanceunknown>=0.5 33  11 no (0.6666667 0.3333333) *
##         31) savings_balanceunknown< 0.5 163  64 yes (0.3926380 0.6073620) *
```

…and plot them:

```
plot(train_rpart$finalModel, main = "Rpart algorithm results")
text(train_rpart$finalModel, cex=0.75)
```

To extract the predictor names from the rpart model that we trained we can use this code below (see the references section: “Introduction to Data Science” by Rafael A. Irizarry“) to see the leafs of the tree:

```
ind <- !(train_rpart$finalModel$frame$var == "<leaf>")
tree_terms <-
  train_rpart$finalModel$frame$var[ind]  %>%
  unique()  %>%
  as.character()
tree_terms
```

```
## [1] "checking_balanceunknown"  "months_loan_duration"
## [3] "checking_balance> 200 DM" "savings_balanceunknown"
```

Let’s see how well the final model performs

```
predict_rpart <- predict(train_rpart, credit_main_test, type = "raw")
accuracy_rpart <- confusionMatrix(predict_rpart,
                                  credit_main_test$default,
                                  positive = "yes")$overall["Accuracy"]
sensitivity_rpart <- sensitivity(predict_rpart,
                                 credit_main_test$default,
                                 positive = "yes")
```

Let’s add these to our risk_models table:

```
risk_models <- rbind(risk_models, list("Rpart Regression Trees (tuned)",
                                       accuracy_rpart,
                                       sensitivity_rpart))
risk_models %>% kable()
```

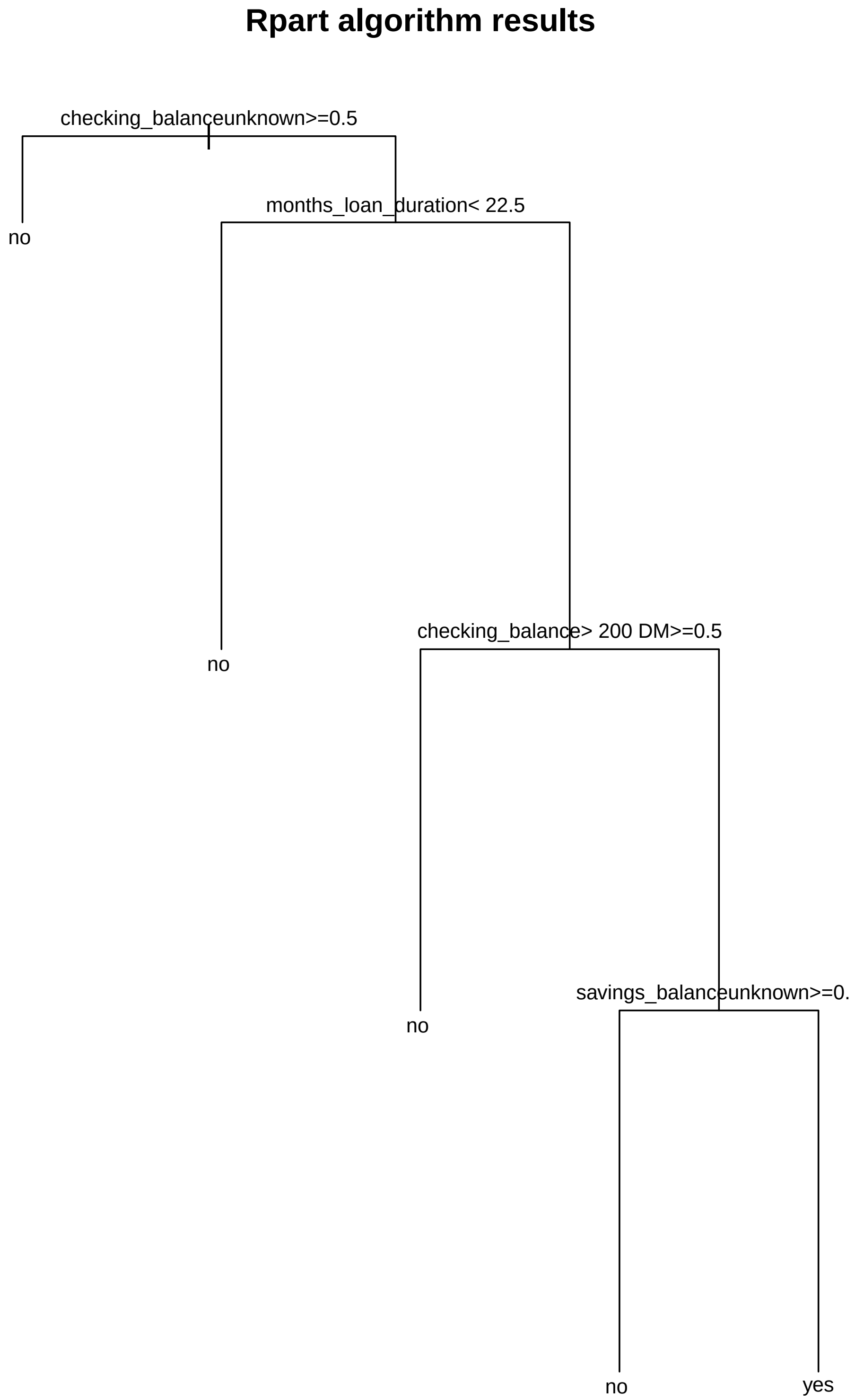

Rpart algorithm results
checking_balanceunknown>=0.5
no
months_loan_duration< 22.5
no
checking_balance> 200 DM>=0.5
no
savings_balanceunknown>=0.
no
yes

Figure 11: Rpart algorithm - final model

	Model	Accuracy	Sensitivity
Accuracy	C5.0 Decision Trees (default)	0.7111111	0.4814815
2	C5.0 Decision Trees (tuned)	0.7222222	0.5185185
3	Rpart Regression Trees (tuned)	0.7444444	0.4814815

With this algorithm we can see a slight increase in accuracy but a lower sensitivity which means that we have managed to correctly predict 48% of the defaulted loans. Can we do better?

Fourth Model: Random Forests (default parameters)

Let's use the caret package implementation of random forests algorithm with default parameters

```
set.seed(123, sample.kind = "Rounding")
train_rf <- train(default ~., credit_main_train, method = "rf")
train_rf
```

```
## Random Forest
##
## 810 samples
##  16 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 810, 810, 810, 810, 810, 810, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7195657  0.1300231
##   18    0.7322546  0.3081377
##   35    0.7272999  0.3101498
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 18.
```

Let's plot the results:

```
ggplot(train_rf, highlight = TRUE) +
  ggtitle("Random forests algorithm - default parameters")
```

And let's now see the final model:

```
train_rf$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = min(param$mtry, ncol(x)))
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 18
##
##         OOB estimate of  error rate: 24.07%
## Confusion matrix:
##      no yes class.error
## no  511  56  0.09876543
## yes 139 104  0.57201646
```

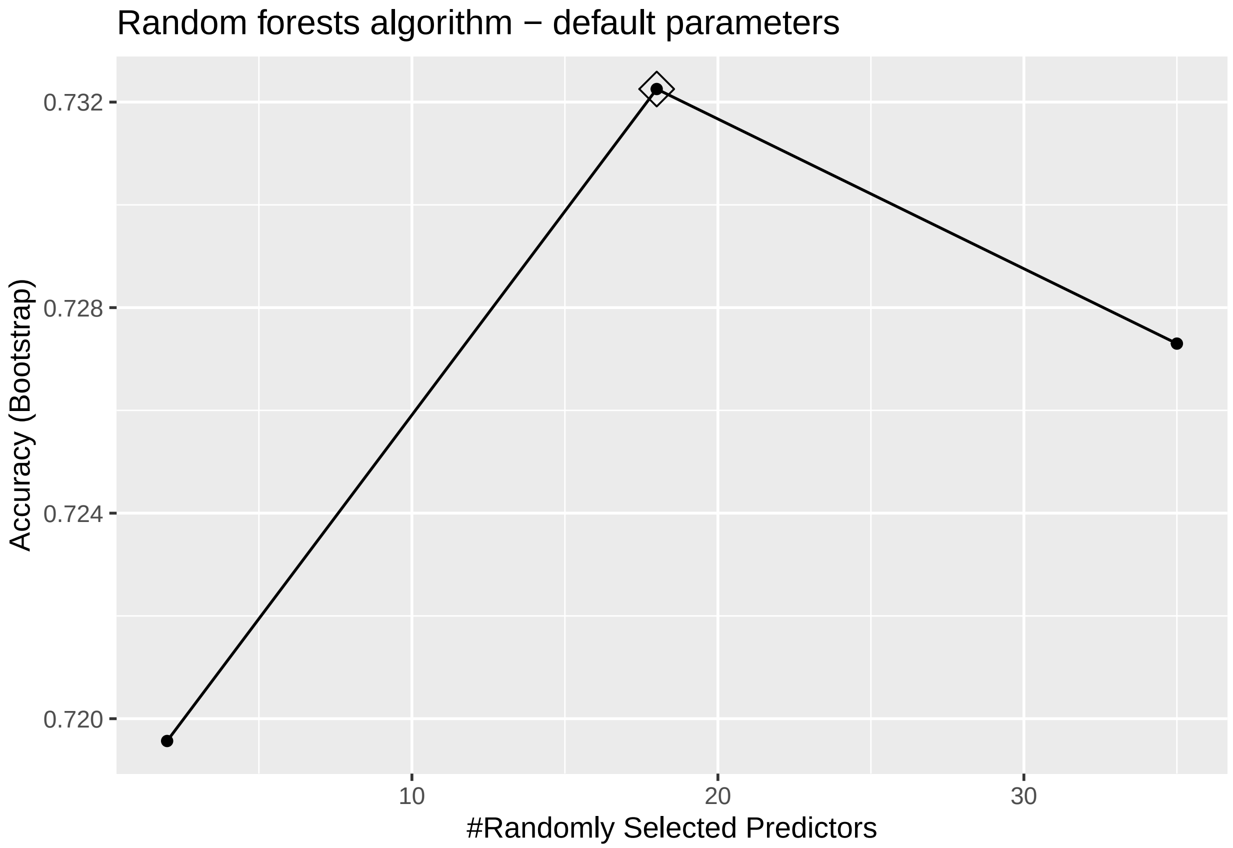


Figure 12: Random forests - default

To find out the importance of each feature we can use again the *varImp* function from the caret package:

```
varImp(train_rf)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                                 Overall
## amount                          100.000
## age                              63.908
## months_loan_duration             57.429
## checking_balanceunknown          47.158
## percent_of_income                22.883
## years_at_residence               22.482
## savings_balanceunknown           12.311
## other_creditnone                 11.546
## existing_loans_count             10.682
## phoneyes                          9.778
## purposecar                        9.356
## credit_historyperfect             8.516
## credit_historyvery good           8.404
## employment_duration1 - 4 years    8.275
## housingown                        7.961
## credit_historygood                7.844
## checking_balance1 - 200 DM        7.662
## purposefurniture/appliances       7.641
## employment_duration> 7 years      7.630
## jobskilled                        7.416
```

And we can see which features are used the most by plotting the variable importance:

```
ggplot(varImp(train_rf)) +
  ggtitle("Variable importance in the Random Forests algorithm")
```

Let’s now see our best tune:

```
train_rf$bestTune
```

```
##   mtry
## 2   18
```

And let’s see how this model performs on the test data:

```
predict_rf <- predict(train_rf, credit_main_test, type = "raw")
```

Let’s calculate the accuracy and sensitivity of our new model:

```
accuracy_rf <- confusionMatrix(predict_rf,
                               credit_main_test$default,
                               positive = "yes")$overall["Accuracy"]
sensitivity_rf <- sensitivity(predict_rf,
                              credit_main_test$default,
                              positive = "yes")
```

Let’s add these to our risk_models table for an easy comparison:

```
risk_models <- rbind(risk_models, list("Random Forests (default)", accuracy_rf, sensitivity_rf))
risk_models %>% kable()
```

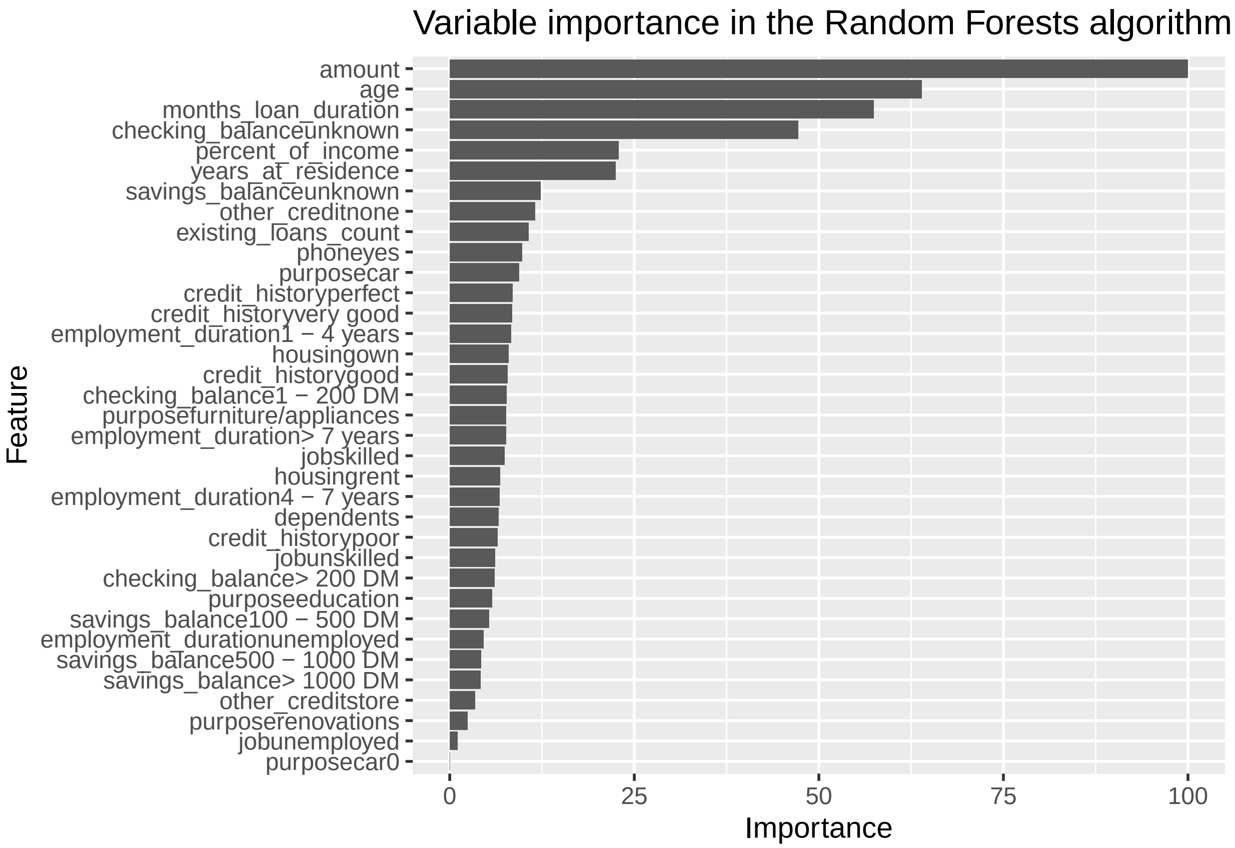


Figure 13: Variable importance in Random Forests algorithm

	Model	Accuracy	Sensitivity
Accuracy	C5.0 Decision Trees (default)	0.7111111	0.4814815
2	C5.0 Decision Trees (tuned)	0.7222222	0.5185185
3	Rpart Regression Trees (tuned)	0.7444444	0.4814815
4	Random Forests (default)	0.7333333	0.5185185

We can now see an improvement in sensitivity despite the fact that our overall accuracy is slightly lower than our previous model. However, this time we are correctly predicting 52% of the defaulted loans. Let's see if we can improve our Random Forests model by tuning the parameters.

Fifth Model: Random Forests (tuned parameters)

Option 1: Tuning the *mtry* parameter only

If we type:

```
modelLookup("rf")
```

```
##   model parameter                         label forReg forClass probModel
## 1    rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
```

we see that the only parameter we can tune in the train function for random forests in the caret package is *mtry*

Let's set the seed first and tune the mtry parameter:

```
set.seed(123, sample.kind = "Rounding")
mtry_grid <- expand.grid(mtry = c(15,18,22,25))
train_rf_tuned <- train(default ~ ., credit_main_train,
                        method = "rf",
                        tuneGrid = mtry_grid)
```

This time, the mtry chosen is 15:

```
train_rf_tuned$bestTune
```

```
##   mtry
## 1   15
```

```
train_rf_tuned
```

```
## Random Forest
##
## 810 samples
##  16 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 810, 810, 810, 810, 810, 810, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   15    0.7365723  0.3150772
##   18    0.7306511  0.3031883
##   22    0.7293580  0.3056062
##   25    0.7292316  0.3076328
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 15.
```

Let's see the performance with the tuned model:

```
predict_rf_tuned <- predict(train_rf_tuned, credit_main_test, type = "raw")
```

Let's calculate the accuracy and sensitivity of our new model:

```
accuracy_rf_tuned <- confusionMatrix(predict_rf_tuned,
                                     credit_main_test$default,
                                     positive = "yes")$overall["Accuracy"]
sensitivity_rf_tuned <- sensitivity(predict_rf_tuned,
                                    credit_main_test$default,
                                    positive = "yes")
```

Let's add these to our risk_models table for an easy comparison:

```
risk_models <- rbind(risk_models, list("Random Forests (tuned)",
                                       accuracy_rf_tuned,
                                       sensitivity_rf_tuned))
risk_models %>% kable()
```

	Model	Accuracy	Sensitivity
Accuracy	C5.0 Decision Trees (default)	0.7111111	0.4814815
2	C5.0 Decision Trees (tuned)	0.7222222	0.5185185
3	Rpart Regression Trees (tuned)	0.7444444	0.4814815
4	Random Forests (default)	0.7333333	0.5185185
5	Random Forests (tuned)	0.7444444	0.4814815

We now obtain a higher accuracy than the previous default model but our sensitivity has dropped. Furthermore, this tuned model performs exactly the same as the Rpart tuned model. Let's now try to optimize our parameter tuning to see if we can do better.

Option 2: Optimizing parameter tuning

Let's set the seed again:

```
set.seed(123, sample.kind = "Rounding")
```

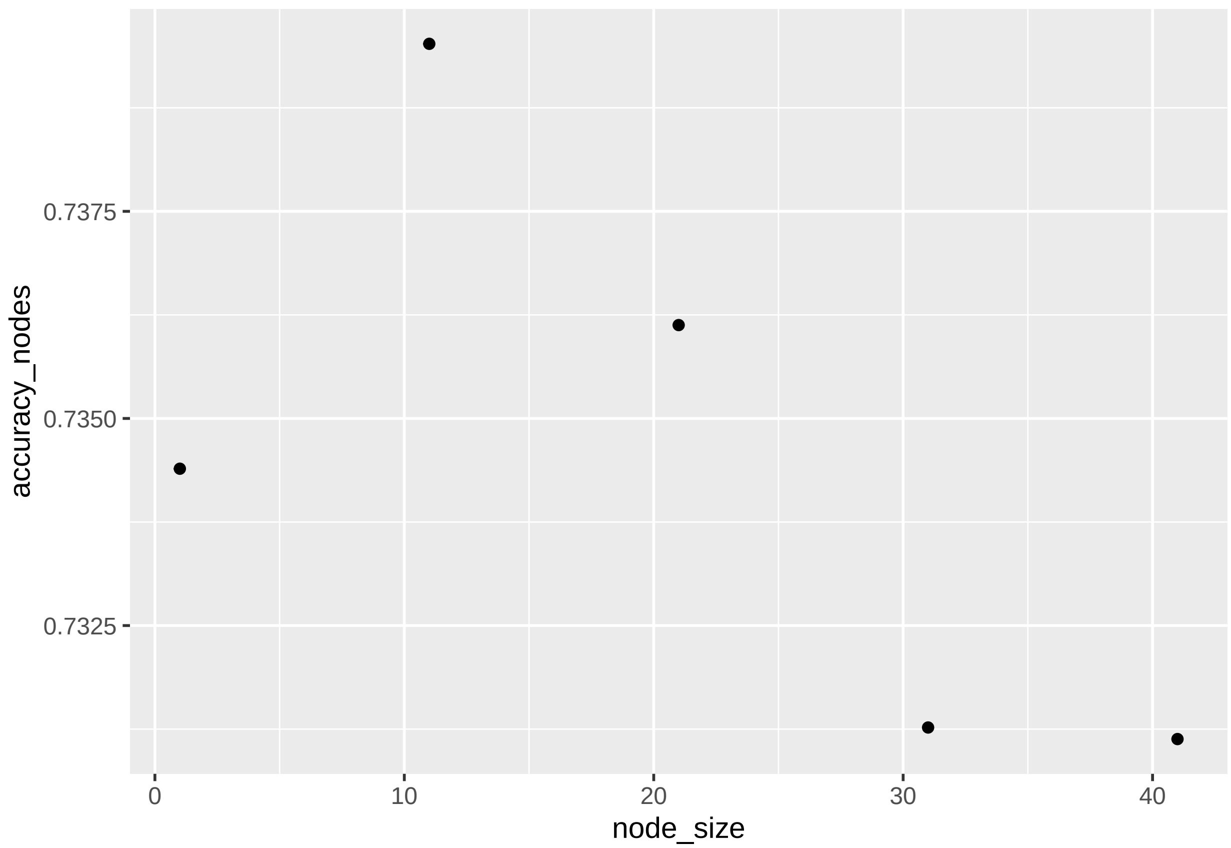
We saw that in the previous random forest model the mtry used was 15 for the final model with the best accuracy. Since in the caret package the only parameter that we can tune is mtry, let's use the *randomForest()* function from the randomForest package instead where we can also tune the minimum number of data points in the nodes of the tree. The higher this number the smoother our estimate can be.

Let's use the caret package to optimize this minimum node size. We'll create a function to calculate the accuracy (reference "Introduction to Data Science" by Rafael A. Irizarry):

```
node_size <- seq(1,50,10)

accuracy_nodes <- sapply(node_size, function(n){
  train(default ~., credit_main_train,
        method = "rf",
        tuneGrid = data.frame(mtry= 15),
        nodesize = node_size)$results$Accuracy
})
```

```
qplot(node_size, accuracy_nodes)
```



We can see the node size for the highest accuracy:

```
node_size[which.max(accuracy_nodes)]
```

```
## [1] 11
```

Let's now apply the optimized node size to the train model:

```
set.seed(123, sample.kind = "Rounding")
train_rf_optz <- randomForest(default ~.,
                              credit_main_train,
                              nodesize = node_size[which.max(accuracy_nodes)])

train_rf_optz
```

```
##
## Call:
##  randomForest(formula = default ~ ., data = credit_main_train,      nodesize = node_size[which.max(ac
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 24.32%
## Confusion matrix:
##      no yes class.error
## no  521  46  0.08112875
## yes 151  92  0.62139918
```

Let’s plot the results:

```
plot(train_rf_optz)
```

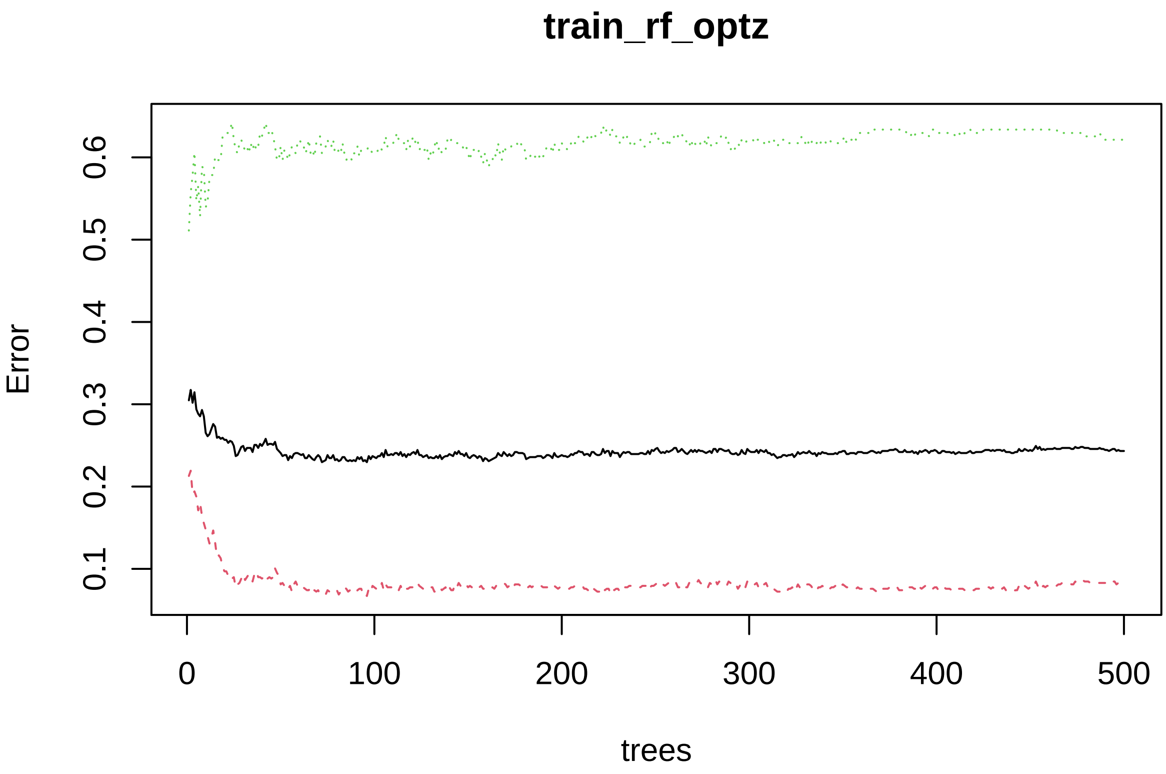


Figure 14: Random Forests model with optimized parameters

Let’s see how well it performs on the test data set:

```
predict_rf_optz <- predict(train_rf_optz, credit_main_test)
accuracy_rf_optz <- confusionMatrix(predict_rf_optz,
                                     credit_main_test$default,
                                     positive = "yes")$overall["Accuracy"]
sensitivity_rf_optz <- sensitivity(predict_rf_optz,
                                   credit_main_test$default,
                                   positive = "yes")
```

Let’s add these results to our *risk_models* table:

```
risk_models <- rbind(risk_models, list("Random Forests (optimized tuning)",
                                       accuracy_rf_optz,
                                       sensitivity_rf_optz))

risk_models %>% kable()
```

	Model	Accuracy	Sensitivity
Accuracy	C5.0 Decision Trees (default)	0.7111111	0.4814815
2	C5.0 Decision Trees (tuned)	0.7222222	0.5185185
3	Rpart Regression Trees (tuned)	0.7444444	0.4814815
4	Random Forests (default)	0.7333333	0.5185185
5	Random Forests (tuned)	0.7444444	0.4814815
6	Random Forests (optimized tuning)	0.7555556	0.4814815

Given that we are interested in a model that has a high overall accuracy with the highest sensitivity the winner is the Random Forests model with default parameters. Therefore, this is the model that we'll use on the *credit_final_holdout_test* dataset.

Results: testing our final model on the final holdout test data set

Let's now test our best model that we have trained so far on the final holdout test dataset that we haven't used so far:

```
predict_final <- predict(train_rf, credit_final_holdout_test)
accuracy_final <- confusionMatrix(predict_final,
                                  credit_final_holdout_test$default,
                                  positive = "yes")$overall["Accuracy"]
accuracy_final
```

```
## Accuracy
##     0.78
```

```
sensitivity_final <- sensitivity(predict_final,
                                 credit_final_holdout_test$default,
                                 positive = "yes")
sensitivity_final
```

```
## [1] 0.5
```

Our model has a 78% accuracy and we have manged to accurately predict 50% of the default loans which is not impressive. Predicting loan defaults from 900 examples seems to be a more challenging task than initially anticipated.

Conclusion and next steps

We saw that using the Random Forests algorithm with the default parameters has provided a higher sensitivity rate than any other model, allowing us to accurately predict 50% of the defaulted loans and offer an overall accuracy of 78%.

Unfortunately, the challenge of correctly predicting the default status of a loan based on only 900 examples has proven greater than anticipated. This may be either because our training dataset was not large enough to properly train our algorithms or perhaps this is a truly difficult challenge in real life.

The advantage of using the Random Forests algorithm over other black box algorithms like kNN for example consists in its transparency since the results of the model can be formulated in plain language.

Next step would be to see if using more sofisticated algorithms would produce higher accuracy and sensitivity even if this means losing the transparency provided by the Random Forests algorithm.

References

- *Introduction to Data Science - Data Analysis and Prediction Algorithms with R* by Rafael A. Irizarry http://rafalab.dfci.harvard.edu/dsbook/
- *Machine Learning with R - Expert Techniques for Predictive Modelling* by Brett Lanz - Third Edition, Packt Publishing https://www.packtpub.com/product/machine-learning-with-r-third-edition/9781788295864#__ga=2.254462029.418584731.1679160506-440441526.1651888047
- The original *german credit* dataset is available on UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data
- The clean *credit.csv* file with 9 attributes is available on Kaggle: https://www.kaggle.com/datasets/uciml/german-credit/download?datasetVersionNumber=1
- The clean *credit.csv* file with 16 attributes is available on GitHub: https://raw.githubusercontent.com/PacktPublishing/Machine-Learning-with-R-Third-Edition/4075e67f7ab26034bc46a8138c08429c2c9e32e8/

Chapter05/credit.csv