

# Loan Default Prediction for Real Estate Companies

Andrea Ramsbacher (18-617-027) & Inderpreet Singh (17-602-947)

December 19th, 2020

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Data</b>	<b>3</b>
2.1	Data Introduction . . . . .	3
2.2	Data Exploration . . . . .	5
2.2.1	Number of Employees . . . . .	5
2.2.2	Number of Jobs Retained . . . . .	6
2.2.3	Number of Jobs Created . . . . .	6
2.3	Data Cleaning . . . . .	8
2.4	Creation of Dummy Variables . . . . .	9
2.5	Data Preprocessing . . . . .	10
<b>3</b>	<b>Classification Models</b>	<b>12</b>
3.1	Model Exploration . . . . .	12
3.1.1	Logistic Regression . . . . .	12
3.1.2	Support Vector Machines . . . . .	16
3.1.3	Random Forest . . . . .	19
3.1.4	Neural network . . . . .	22
3.1.5	Extreme Gradient Boosting . . . . .	25
3.2	Stacked Model . . . . .	28
3.2.1	Model Development . . . . .	28
<b>4</b>	<b>Conclusion</b>	<b>32</b>
<b>5</b>	<b>References</b>	<b>33</b>

# 1 Introduction

Throughout the years, the loan guarantees programme of the U.S. Small Business Administration (SBA) bred dozens of success stories of start-ups growing into tremendous companies, such as FedEx and Apple to only name a few. However, behind this shiny facade, there have also been many occurrences of small businesses ultimately defaulting on their SBA-guaranteed loans and hence never repaying their debt.

This project focuses on building a model which analyses loan guarantees applicants and classifies them as either prone to defaulting (i.e. failing to pay back their loan) or safe to extend a loan to. The matter at hand is first tackled through a brief data introduction and data exploration focused on real estate companies in order to better grasp the provided dataset. Then, the dataset is cleaned and supplemented with a few useful additional dummy variables before preprocessing it for the subsequent classification. Ultimately, various models are investigated and a stacked model is then created to capitalise on the strengths of each individual model and give rise to a powerful classifier for loan default detection.

## 2 Data

Required Libraries

```
rm(list=ls( ))
cat("\014")
set.seed(12345)
library(data.table)
library(tidyverse)
library(lubridate)
library(caret)
library(caretEnsemble)
```

### 2.1 Data Introduction

The dataset is provided for the competition ‘Should This Loan be Approved or Denied?’ on Kaggle and stems out of the U.S. Small Business Administration (SBA). The SBA was founded in 1953 on the principle of promoting and assisting small enterprises in the U.S. credit market. The latter companies historically have been a primary source of job creation in the US. Hence, fostering small business formation and growth has significant social and economic benefits as it contributes to creating job opportunities and reducing unemployment.

The dataset contains various pieces of information, the table below introduces a few key variables.

Variable Name	Description
NAICS	North American industry classification system code
DisbursementGross	Amount disbursed
NoEmp	Number of people employed in the company
RevLineCr	Whether or not the loan is part of a revolving line of credit
NewExist	Whether the company is a new or an existing company
Term	Loan term in months
UrbanRural	Type of environment (Urban, Rural or undefined location)
MIS_Status	Whether the loan has been paid in full (PIF) or charged off (CHGOFF)

First, we import the dataset as a data table.

```
SBAnational <- fread('SBAnational.csv')
```

Using the NAICS of the real estate industry, we then only keep the rows for the RERL companies as it is the industry on which this analysis focuses.

```
as.numeric(as.character(SBAnational$NAICS))
data_RERL <- setDT(SBAnational)[substr(SBAnational$NAICS, 1,2) == 53]
```

One then checks whether there are any NA rows in the dataset.

```
colSums(is.na(data_RERL))
```

```
##      LoanNr_ChkDgt      Name      City      State
##           0           0           0           0
##           Zip           Bank      BankState      NAICS
##           0           0           0           0
##      ApprovalDate      ApprovalFY      Term      NoEmp
##           0           0           0           0
##           NewExist      CreateJob      RetainedJob      FranchiseCode
##           2           0           0           0
##           UrbanRural      RevLineCr      LowDoc      ChgOffDate
##           0           0           0           0
##      DisbursementDate      DisbursementGross      BalanceGross      MIS_Status
##           0           0           0           0
##           ChgOffPrinGr      GrAppv      SBA_Appv
##           0           0           0
```

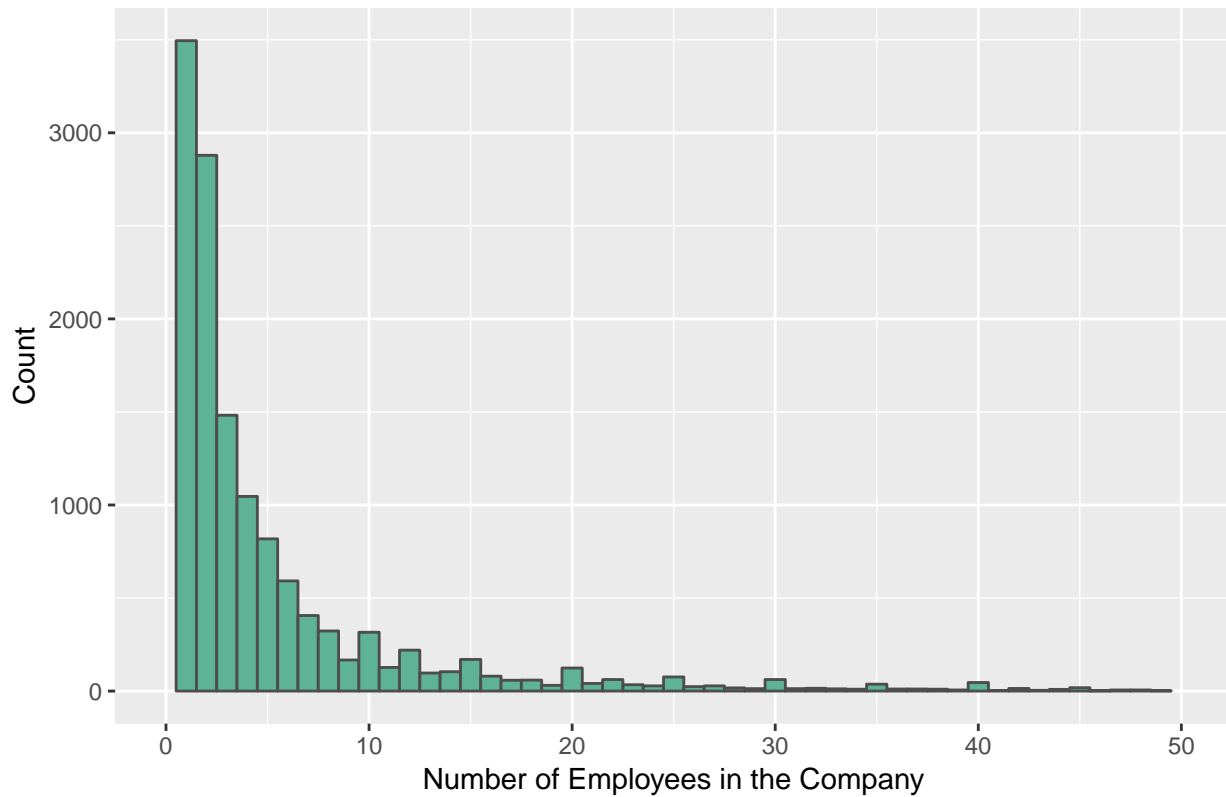
## 2.2 Data Exploration

### 2.2.1 Number of Employees

It is interesting to notice that while the number of employees in the companies investigated ranges from 1 to 2000, the median value is 3. Hence, one may affirm that the enterprises inquiring SBA guaranteed loans are on average of fairly modest size. Furthermore, when looking at the distribution of the number of employees per company, one observes that the count of companies diminishes dramatically as the number of employees increases. Most companies count between 1 and 5 employees.

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.000	1.000	3.000	7.367	6.000	2000.000

Distribution of the Number of Employees

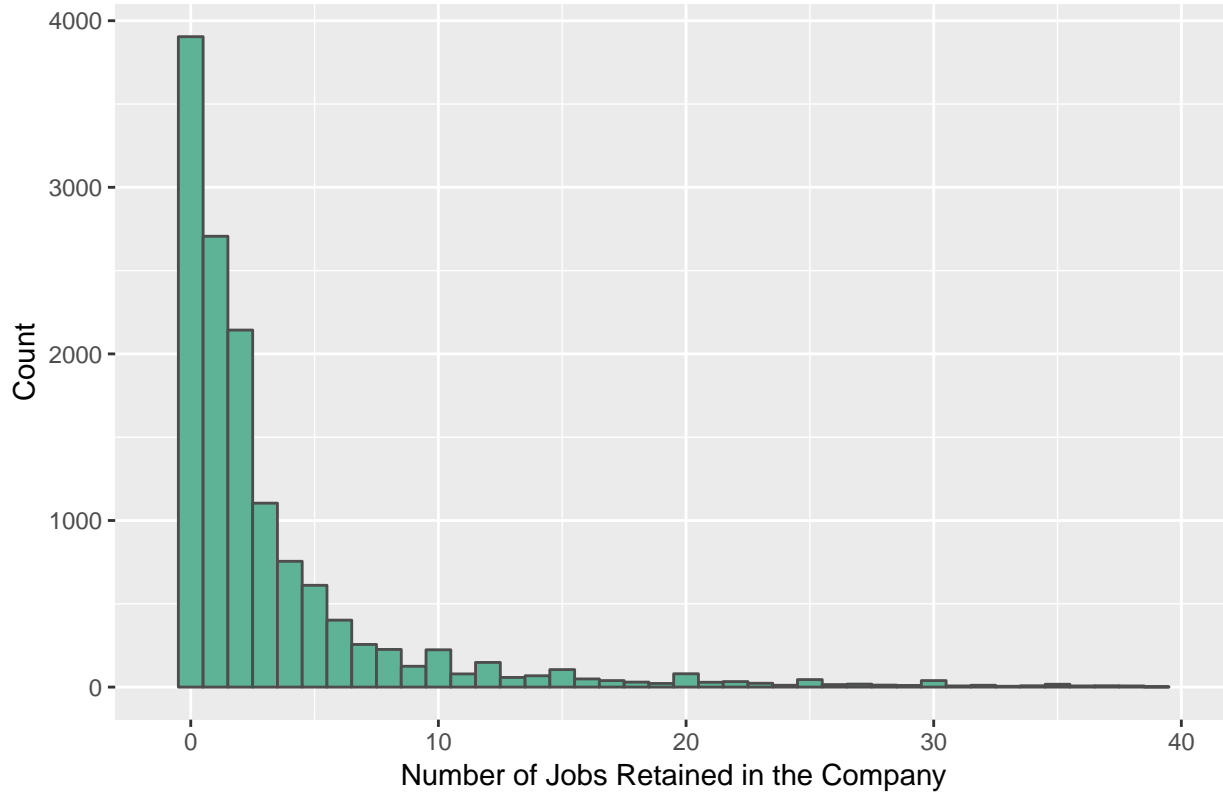


### 2.2.2 Number of Jobs Retained

In terms of jobs retained and jobs created, we will again focus on the median value rather than the mean as it is more meaningful with skewed data. We see that the median number of jobs retained lies at 2, whereas the median number of jobs created is zero. Therefore, one may think that loans contracted by real estate companies generally do not work towards expanding the company, but maybe rather towards maintaining the existing structure.

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.000	0.000	2.000	4.359	4.000	535.000

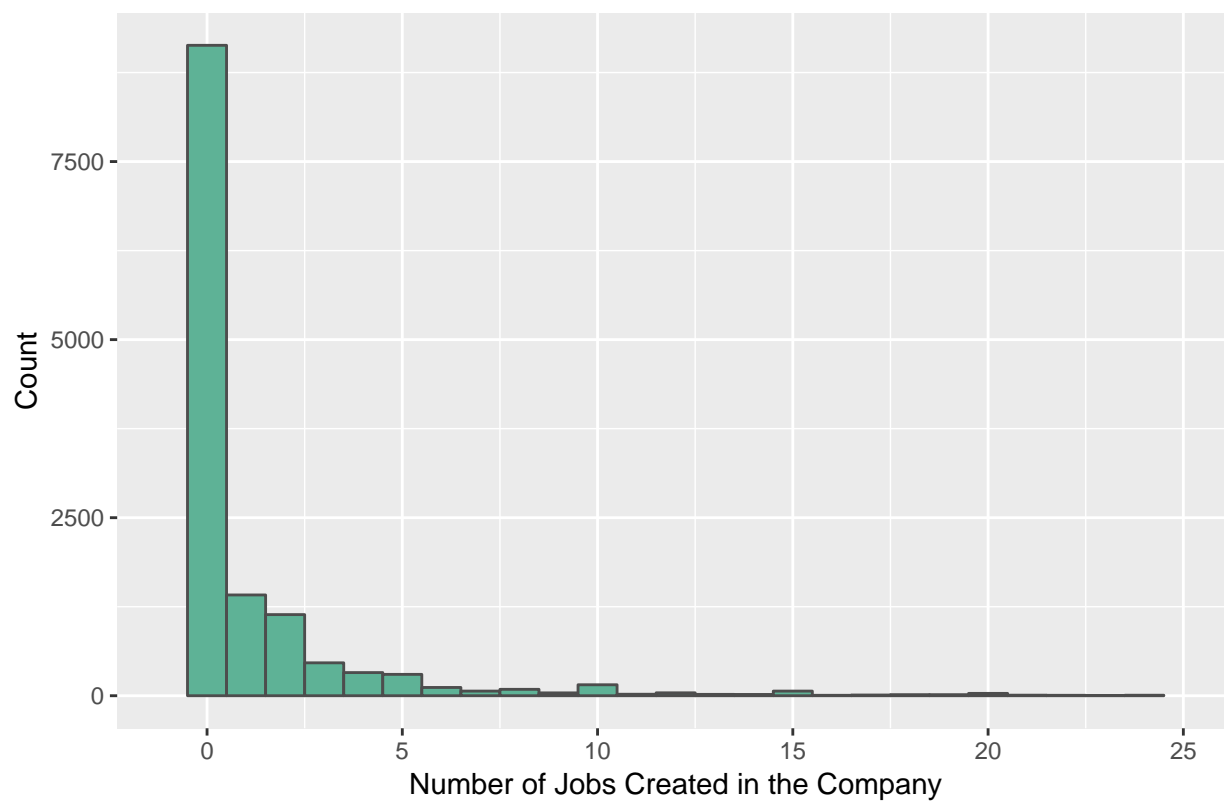
Distribution of the Number of Jobs Retained



### 2.2.3 Number of Jobs Created

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.000	0.000	0.000	1.618	1.000	452.000

Distribution of the Number of Jobs Created



## 2.3 Data Cleaning

The format of the amounts in US dollars needs to be amended in order to be used in the analysis. '\$' signs, commas and dots are hereafter removed.

```
data_RERL$DisbursementGross <- as.numeric(gsub('[$,]', '', data_RERL$DisbursementGross))
data_RERL$BalanceGross <- as.numeric(gsub('[$,]', '', data_RERL$BalanceGross))
data_RERL$ChgOffPrinGr <- as.numeric(gsub('[$,]', '', data_RERL$ChgOffPrinGr))
data_RERL$GrAppv <- as.numeric(gsub('[$,]', '', data_RERL$GrAppv))
data_RERL$SBA_Appv <- as.numeric(gsub('[$,]', '', data_RERL$SBA_Appv))
```

The format of calendar (dates) variables in the data are corrected.

```
ApprovalDate <- format(strptime(data_RERL$ApprovalDate,
                                format = '%d-%b-%y'))
ApprovalDate <- data.table(ApprovalDate)

ChgOffDate <- format(strptime(data_RERL$ChgOffDate,
                              format = '%d-%b-%y'))
ChgOffDate <- data.table(ChgOffDate)

DisbursementDate <- format(strptime(data_RERL$DisbursementDate,
                                     format = '%d-%b-%y'))
DisbursementDate <- data.table(DisbursementDate)

data_RERL$ApprovalDate <- ApprovalDate
data_RERL$DisbursementDate <- DisbursementDate
data_RERL$ChgOffDate <- ChgOffDate
```

We then remove all rows which do not contain a value for the MIS\_status.

```
data_RERL <- data_RERL[data_RERL$MIS_Status == 'P I F' | data_RERL$MIS_Status == 'CHGOFF', ]
```

This enables to have existing factor name when we create the ensemble model.

```
data_RERL[data_RERL$MIS_Status == 'P I F', c("MIS_Status")] = 'PIF'
```

One then gets rid of all the incoherent values for NewExist.

```
data_RERL <- data_RERL[data_RERL$NewExist == 1 | data_RERL$NewExist == 2, ]
```

The relevant variables are converted into factors.

```
data_RERL$State <- as.factor(data_RERL$State)
data_RERL$NewExist <- as.factor(data_RERL$NewExist)
data_RERL$UrbanRural <- as.factor(data_RERL$UrbanRural)
data_RERL$MIS_Status <- as.factor(data_RERL$MIS_Status)
```

As most loans last 5 years or longer, default information is not systematically available for loans that have been emitted within 5 years of the most recent date included in the dataset. Therefore, only rows with disbursement date no more recent than 2010 are kept in the dataset.

```
data_RERL <- data_RERL[DisbursementDate <= 2011-01-01, ]
```

## 2.4 Creation of Dummy Variables

One needs to create a dummy variable identifying real estate investments. In order to do so, loans with a term of over 240 months are selected, that is, when the term is longer than 20 years, it is considered RE. The dummy variable shows a one when it is RE and 0 otherwise.

```
RealEstate <- ifelse(data_RERL$Term >= 240, 1, 0)
data_RERL <- data_RERL %>%
  mutate(RealEstate)
```

Furthermore, one also needs to create a variable showing the portion of the loan guaranteed by the SBA.

```
GuaranteedPortion <- data_RERL$SBA_Appv/data_RERL$GrAppv
data_RERL <- data_RERL %>%
  mutate(GuaranteedPortion)
```

Last, one creates a variable depicting whether or not the loan was active during a recession period (as this has a significant impact on the debt default payment). Moderate recessions have a weight of 1, whereas the great financial crisis of 2007 has a weight of 2.

```
Recession <- ifelse(data_RERL$ApprovalDate >= "2007-12-07" &
  data_RERL$DisbursementDate <= "2009-06-31", 2,
  ifelse(data_RERL$ApprovalDate >= "1987-10-18" &
    data_RERL$DisbursementDate <= "1988-04-15", 1,
    ifelse(data_RERL$ApprovalDate >= "1990-07-01" &
      data_RERL$DisbursementDate <= "1991-03-20", 1,
      ifelse(data_RERL$ApprovalDate >= "2001-03-01" &
        data_RERL$DisbursementDate <= "2001-11-20", 1, 0))))
data_RERL <- data_RERL %>%
  mutate(Recession)
```

We now rename and convert into factors the explained variable ‘Default’.

```
data_RERL$Default <- as.factor(data_RERL$MIS_Status)
```

We convert relevant dummy variables into factors.

```
data_RERL$RealEstate <- as.factor(data_RERL$RealEstate)
data_RERL$Recession <- as.factor(data_RERL$Recession)
data_RERL$RevLineCr <- as.factor(data_RERL$RevLineCr)
```

## 2.5 Data Preprocessing

We remove irrelevant variables from the dataset.

```
data_RERL$State <- NULL
data_RERL$LoanNr_ChkDgt <- NULL
data_RERL$Name <- NULL
data_RERL$City <- NULL
data_RERL$Zip <- NULL
data_RERL$Bank <- NULL
data_RERL$BankState <- NULL
data_RERL$NAICS <- NULL
data_RERL$ApprovalDate <- NULL
data_RERL$ApprovalFY <- NULL
data_RERL$FranchiseCode <- NULL
data_RERL$LowDoc <- NULL
data_RERL$ChgOffDate <- NULL
data_RERL$DisbursementDate <- NULL
data_RERL$BalanceGross <- NULL
data_RERL$MIS_Status <- NULL
data_RERL$ChgOffPrinGr <- NULL
data_RERL$GrAppv <- NULL
data_RERL$SBA_Appv <- NULL
```

The data is now further preprocessed by standardising and normalising the variables in order to enable a clean one hot encoding. More precisely “Center” transformation simply subtracts the mean of the covariate in question (to center the average at 0), while the “scale” transformation will divide by the standard deviation. Thus, if the distribution of the random variable has a normal distribution, the latter would follow a standard normal distribution  $X \sim N(0,1)$  and test statistics are simplified. However, it is practically never the case that a RV follows a perfect (if not at all) normal shape. This is where the “BoxCox” transformation proves useful. To assure that the ‘Classical Linear Model’ assumptions hold, Box and Cox (1964) developed a model that power-transforms almost any RV distributions to normal. YeoJohnson transformation is similar to Box-Cox but allows for 0 and negative values, and hence will be preferred here.

```
processed = preprocess(data_RERL, method=c("scale", "center", "YeoJohnson"))
data_RERL = predict(processed, newdata = data_RERL)
```

We then range all features so that they range from 0 to 1.

```
range = preprocess(data_RERL, method=c("range"))
data_RERL = predict(range, newdata = data_RERL)
```

The variable ‘Default’ is stored as it will be needed after one-hot encoding the dataset.

```
Default_copy <- data_RERL$Default
```

The dataset is now one-hot encoded using the DummyVars function. This transformation is particularly necessary for models that require continuous numeric variables for training. For instance, a neural network needs numeric values to evaluate and minimise its loss function with the backpropagation algorithm.

```
dummies <- dummyVars(Default ~ ., data = data_RERL, fullRank = TRUE)
data_mat <- predict(dummies, newdata = data_RERL)
data_RERL <- data.frame(data_mat)

data_RERL$Default <- Default_copy

str(data_RERL)
```

```
## 'data.frame':    12419 obs. of  17 variables:
## $ Term           : num  0.554 0.532 0.574 0.453 0.539 ...
## $ NoEmp           : num  0.547 0.642 0.466 0.326 0.326 ...
## $ NewExist.2      : num  0 0 1 0 0 0 0 0 0 0 ...
## $ CreateJob       : num  0 0.949 0 0 0 ...
## $ RetainedJob     : num  0 0.555 0 0 0 ...
## $ UrbanRural.1    : num  1 0 0 0 0 0 0 0 1 0 ...
## $ UrbanRural.2    : num  0 1 1 0 0 1 0 0 0 1 ...
## $ RevLineCr.0     : num  0 0 0 0 0 0 0 0 1 0 ...
## $ RevLineCr.N     : num  1 0 1 0 0 0 0 0 0 1 ...
## $ RevLineCr.T     : num  0 0 0 0 0 0 0 0 0 0 ...
## $ RevLineCr.Y     : num  0 1 0 1 1 1 1 1 0 0 ...
## $ DisbursementGross: num  0.398 0.47 0.559 0.334 0.321 ...
## $ RealEstate.1    : num  0 0 0 0 0 0 0 0 1 0 ...
## $ GuaranteedPortion: num  0.837 0.457 0.457 0.457 0.457 ...
## $ Recession.1     : num  0 0 0 1 0 0 1 0 0 0 ...
## $ Recession.2     : num  0 0 0 0 0 0 0 0 0 0 ...
## $ Default         : Factor w/ 2 levels "CHGOFF","PIF": 2 1 2 2 2 2 2 2 2 2 ...
```

The explained variable 'Default' is moved from the last to the first column for convenience of use in the subsequent analyses.

```
data_RERL <- data_RERL[,c(ncol(data_RERL),1:(ncol(data_RERL)-1))]
```

We proceed to the train and test data separation. The sample function was chosen to separate 75% of the total data set to the train dataset and the remaining to the test one.

```
n <- nrow(data_RERL)
index = sample(1:n, size = round(n*0.75), replace = FALSE)
train_data_RERL <- data_RERL[index,]
test_data_RERL <- data_RERL[-index,]
```

## 3 Classification Models

### 3.1 Model Exploration

Hereafter, various models are explored in order to select the models, which will be part of the final stacked ensemble.

#### 3.1.1 Logistic Regression

##### 3.1.1.1 Model Description

The logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. In regression analysis, the logistic regression is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values (labeled 0 and 1). In the logistic model, the log-odds (the logarithm of the odds) for the value labeled “1” is a linear combination of one or more independent variables. The independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled “1” can vary between 0 (certainly the value “0”) and 1 (certainly the value “1”), hence the labeling; the function that converts log-odds to probability is the logistic function, from which the model inherits its name.

We now evaluate a simple logistic regression using the Caret package. We use the logistic regression as an indicator for the theoretical default probabilities that are obtained from the prediction function. By solving for each betas of the log odds equation, we induce intuitive properties on the impact of each variables on (logg odds) default probabilities.

##### 3.1.1.2 Model Training and Testing

```
fitControl <- trainControl(method = "none")
```

```
logReg <- train(Default ~ .,  
               data = train_data_RERL,  
               method = "glm",  
               family = binomial,  
               trControl = fitControl)
```

```
options(scipen=999)
```

```
summary(logReg)
```

```
##  
## Call:  
## NULL  
##  
## Deviance Residuals:  
##      Min       1Q   Median       3Q      Max   
## -3.2687  -0.5803   0.3246   0.6798   3.3858   
##  
## Coefficients:  
##              Estimate Std. Error z value      Pr(>|z|)        
## (Intercept)    7.637825  177.636448   0.043      0.965704        
## Term          12.327579   0.347244  35.501 < 0.0000000000000002 ***  
## NoEmp         1.865759   0.219486   8.501 < 0.0000000000000002 ***  
## NewExist.2     0.222460   0.066823   3.329      0.000871 ***
```

```
## CreateJob          0.005499   0.078160   0.070                0.943910
## RetainedJob        -1.594658   0.182809  -8.723 < 0.0000000000000002 ***
## UrbanRural.1       -2.013374   0.151394 -13.299 < 0.0000000000000002 ***
## UrbanRural.2       -1.593883   0.163083  -9.773 < 0.0000000000000002 ***
## RevLineCr.0        -10.960850  177.636273 -0.062                0.950799
## RevLineCr.N        -11.919207  177.636258 -0.067                0.946503
## RevLineCr.T        -12.494435  177.636336 -0.070                0.943925
## RevLineCr.Y        -11.353283  177.636274 -0.064                0.949039
## DisbursementGross  -0.682211   0.192800  -3.538                0.000402 ***
## RealEstate.1       -1.564552   0.209784  -7.458   0.00000000000000879 ***
## GuaranteedPortion  -1.510485   0.266469  -5.669   0.0000000144033132 ***
## Recession.1         0.856593   0.242869   3.527                0.000420 ***
## Recession.2        -0.393151   0.108270  -3.631                0.000282 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##    Null deviance: 11473.8  on 9313  degrees of freedom
## Residual deviance:  7819.5  on 9297  degrees of freedom
## AIC: 7853.5
##
## Number of Fisher Scoring iterations: 13
```

```
varImp(logReg)
```

```
## glm variable importance
##
##              Overall
## Term          100.000000
## UrbanRural.1   37.351435
## UrbanRural.2   27.403786
## RetainedJob    24.439876
## NoEmp          23.812029
## RealEstate.1   20.869963
## GuaranteedPortion 15.820821
## Recession.2    10.072135
## DisbursementGross 9.810347
## Recession.1     9.778007
## NewExist.2      9.219573
## CreateJob       0.024414
## RevLineCr.T     0.024361
## RevLineCr.N     0.015223
## RevLineCr.Y     0.006234
## RevLineCr.0     0.000000
```

Predictions:

```
prediction.train.log = predict(logReg, newdata = train_data_RERL)
prediction.test.log = predict(logReg, newdata = test_data_RERL)
```

### 3.1.1.3 Logistic Regression Performance on the Training Data

```
confusionMatrix(data = prediction.train.log, reference = train_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF    1724  453
##   PIF       1127 6010
##
##           Accuracy : 0.8304
##           95% CI : (0.8226, 0.8379)
##   No Information Rate : 0.6939
##   P-Value [Acc > NIR] : < 0.00000000000000022
##
##           Kappa : 0.5724
##
##   McNemar's Test P-Value : < 0.00000000000000022
##
##           Sensitivity : 0.6047
##           Specificity : 0.9299
##   Pos Pred Value : 0.7919
##   Neg Pred Value : 0.8421
##   Prevalence : 0.3061
##   Detection Rate : 0.1851
##   Detection Prevalence : 0.2337
##   Balanced Accuracy : 0.7673
##
##   'Positive' Class : CHGOFF
##
```

### 3.1.1.4 Logistic Regression Performance on the Testing Data

```
confusionMatrix(data = prediction.test.log, reference = test_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF    568  147
##   PIF       342 2048
##
##           Accuracy : 0.8425
##           95% CI : (0.8292, 0.8552)
##   No Information Rate : 0.7069
##   P-Value [Acc > NIR] : < 0.00000000000000022
##
##           Kappa : 0.5945
##
##   McNemar's Test P-Value : < 0.00000000000000022
##
##           Sensitivity : 0.6242
##           Specificity : 0.9330
##   Pos Pred Value : 0.7944
```

```

##          Neg Pred Value : 0.8569
##          Prevalence : 0.2931
##          Detection Rate : 0.1829
##    Detection Prevalence : 0.2303
##          Balanced Accuracy : 0.7786
##
##          'Positive' Class : CHGOFF
##

```

Two important features are considered in our analysis, namely sensitivity and specificity. Sensitivity describes the process of checking whether the estimated effects and statistical significance of key explanatory variables are sensitive to inclusion of other explanatory variables (Wooldridge, 2015, p.767). In our data, it translates in truly identifying defaulting companies which will not pay back their debt (true positive examples). On the other hand, specificity measures the correct number of negative predictions, therefore truly identifying the firms that will pay their loans back as per their agreement with the SBA (true negative examples). Last, the accuracy demonstrates the total number of correct predictions of the model (basic evaluation measures from the confusion matrix, s.d.).

In respect of those various metrics, one can observe that the sensitivity of our logistic regression in both the training and testing data is around 60%, which suggests the method is only moderately successful and that it is likely not the best model for true positive predictions. Specificity, however, fares way better as it achieves a value of 92.99% in the trained data and 93.3% in the testing data. The accuracy of the model, which reflects the total number of correct predictions, reaches 83%. Overall, the logistic regression is a good model for specificity, that is in terms of predicting which firms will pay back their loans, but it performs poorly in terms of sensitivity.

### 3.1.2 Support Vector Machines

#### 3.1.2.1 Model Description

Support-vector machines (SVMs) are a supervised learning model with associated learning algorithms that analyse data for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM maps training examples to points in space to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

We now evaluate a linear SVM using the Caret package.

#### 3.1.2.2 Model Training and Testing

```
svmModel <- train(Default ~ ., data = train_data_RERL, method = "svmLinear",  
                  trControl = fitControl)
```

```
options(scipen=999)
```

```
summary(svmModel)
```

```
## Length Class Mode  
##      1  ksvm   S4
```

```
varImp(svmModel)
```

```
## ROC curve variable importance  
##  
##              Importance  
## Term                100.0000  
## GuaranteedPortion    39.6990  
## DisbursementGross    29.6311  
## RevLineCr.0          25.6483  
## RetainedJob          25.3425  
## UrbanRural.1         23.0039  
## NoEmp                22.9677  
## RealEstate.1         22.5196  
## RevLineCr.Y          21.7392  
## RevLineCr.T          4.3882  
## NewExist.2           3.3802  
## Recession.2           3.3257  
## UrbanRural.2         1.9677  
## Recession.1          1.9317  
## CreateJob            0.5415  
## RevLineCr.N          0.0000
```

Predictions:

```
prediction.train.svm = predict(svmModel, newdata = train_data_RERL)  
prediction.test.svm = predict(svmModel, newdata = test_data_RERL)
```

### 3.1.2.3 SVM Performance on the Training Data

```
confusionMatrix(data = prediction.train.svm, reference = train_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF    1865  440
##   PIF        986 6023
##
##           Accuracy : 0.8469
##           95% CI : (0.8394, 0.8542)
##   No Information Rate : 0.6939
##   P-Value [Acc > NIR] : < 0.00000000000000022
##
##           Kappa : 0.6192
##
## Mcnemar's Test P-Value : < 0.00000000000000022
##
##           Sensitivity : 0.6542
##           Specificity : 0.9319
##   Pos Pred Value : 0.8091
##   Neg Pred Value : 0.8593
##   Prevalence : 0.3061
##   Detection Rate : 0.2002
##   Detection Prevalence : 0.2475
##   Balanced Accuracy : 0.7930
##
##   'Positive' Class : CHGOFF
##
```

### 3.1.2.4 SVM Performance on the Testing Data

```
confusionMatrix(data = prediction.test.svm, reference = test_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF    630  146
##   PIF       280 2049
##
##           Accuracy : 0.8628
##           95% CI : (0.8502, 0.8747)
##   No Information Rate : 0.7069
##   P-Value [Acc > NIR] : < 0.00000000000000022
##
##           Kappa : 0.654
##
## Mcnemar's Test P-Value : 0.0000000001165
##
##           Sensitivity : 0.6923
##           Specificity : 0.9335
##   Pos Pred Value : 0.8119
```

```
##          Neg Pred Value : 0.8798
##          Prevalence : 0.2931
##          Detection Rate : 0.2029
##    Detection Prevalence : 0.2499
##          Balanced Accuracy : 0.8129
##
##          'Positive' Class : CHGOFF
##
```

The SVM model achieves slightly better performances than the logistic regression in terms of prediction. One should however note that the sensitivity is diverging by 3.81% between the trained and the testing data. Indeed, the sensitivity reaches 65.42% on the trained data and 69.23% on the testing data, which is still better than the logistic regression, but might suggest that some parameters might be overfitted (just as a model may be overfitted) due to the small size of the dataset. The specificity ranges between 93.19% and 93.35%, which is also better than the previous model. Additionally, the accuracy is slightly higher with 84.69% and 86.28%. Overall, the SVM model is superior in all metrics, i.e. sensitivity, specificity and accuracy.

### 3.1.3 Random Forest

#### 3.1.3.1 Model Description

Random forests are an ensemble learning method for classification (inter alia, the method may also be used for regression tasks). The algorithm constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes (in the setting of a classification task). The key benefit of using random decision forests resides in the fact that it corrects for decision trees' habit of overfitting to their training set.

We now evaluate a random forest using the Caret package.

#### 3.1.3.2 Model Training and Testing

```
RandomForest <- train(Default ~ ., data = train_data_RERL, method = "rf",  
                      trControl = fitControl)  
options(scipen=999)
```

```
summary(RandomForest)
```

##	Length	Class	Mode
## call	4	-none-	call
## type	1	-none-	character
## predicted	9314	factor	numeric
## err.rate	1500	-none-	numeric
## confusion	6	-none-	numeric
## votes	18628	matrix	numeric
## oob.times	9314	-none-	numeric
## classes	2	-none-	character
## importance	16	-none-	numeric
## importanceSD	0	-none-	NULL
## localImportance	0	-none-	NULL
## proximity	0	-none-	NULL
## ntree	1	-none-	numeric
## mtry	1	-none-	numeric
## forest	14	-none-	list
## y	9314	factor	numeric
## test	0	-none-	NULL
## inbag	0	-none-	NULL
## xNames	16	-none-	character
## problemType	1	-none-	character
## tuneValue	1	data.frame	list
## obsLevels	2	-none-	character
## param	0	-none-	list

```
varImp(RandomForest)
```

```
## rf variable importance  
##  
## Overall  
## Term 100.00000  
## DisbursementGross 15.96129  
## RetainedJob 8.55120  
## GuaranteedPortion 6.30999  
## NoEmp 5.70374  
## RevLineCr.0 3.46794  
## CreateJob 3.34093
```

```
## UrbanRural.1      2.46386
## RealEstate.1      1.26795
## NewExist.2        1.26483
## RevLineCr.N       0.81752
## RevLineCr.Y       0.66383
## UrbanRural.2      0.60973
## Recession.2       0.59099
## Recession.1       0.01104
## RevLineCr.T       0.00000
```

Predictions:

```
prediction.train.rf = predict(RandomForest, newdata = train_data_RERL)
prediction.test.rf = predict(RandomForest, newdata = test_data_RERL)
```

### 3.1.3.3 Random Forest Performance on the Training Data

```
confusionMatrix(prediction.train.rf, train_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction CHGOFF PIF
##    CHGOFF    2816   49
##    PIF         35 6414
##
##              Accuracy : 0.991
##              95% CI : (0.9888, 0.9928)
##    No Information Rate : 0.6939
##    P-Value [Acc > NIR] : <0.0000000000000002
##
##              Kappa : 0.9788
##
## Mcnemar's Test P-Value : 0.1561
##
##              Sensitivity : 0.9877
##              Specificity : 0.9924
##              Pos Pred Value : 0.9829
##              Neg Pred Value : 0.9946
##              Prevalence : 0.3061
##              Detection Rate : 0.3023
##    Detection Prevalence : 0.3076
##              Balanced Accuracy : 0.9901
##
##              'Positive' Class : CHGOFF
##
```

### 3.1.3.4 Random Forest Performance on the Testing Data

```
confusionMatrix(prediction.test.rf, test_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction CHGOFF PIF
```

```

##      CHGOFF      774  131
##      PIF        136 2064
##
##              Accuracy : 0.914
##              95% CI  : (0.9036, 0.9236)
##      No Information Rate : 0.7069
##      P-Value [Acc > NIR] : <0.0000000000000002
##
##              Kappa : 0.7921
##
##      Mcnemar's Test P-Value : 0.8066
##
##              Sensitivity : 0.8505
##              Specificity : 0.9403
##              Pos Pred Value : 0.8552
##              Neg Pred Value : 0.9382
##              Prevalence : 0.2931
##              Detection Rate : 0.2493
##      Detection Prevalence : 0.2915
##              Balanced Accuracy : 0.8954
##
##      'Positive' Class : CHGOFF
##

```

The sensitivity of the random forest model is considerably higher than that of the two previous models. It achieves a sensitivity of 98.77% in the training data and 85.05% in the testing data. Hence, this might suggest that the model is overfitting on the training data. In terms of specificity, the difference is minor and the value achieved is well above that of the two other models with 99.24% (training data) and 94.03% (testing data). The accuracy also mounts to 99.1% (training data) and 91.4% (testing data), reflecting the disparities in sensitivity already observed above. All in all, the performances are superior in all respects, but testing data performances are systematically lower than those obtained on the training data. While the model might overfit to some extent on the training data, it nonetheless performs better and is more accurate than both the logistic regression and the SVM model.

### 3.1.4 Neural network

#### 3.1.4.1 Model Description

A neural network is a multivariate regression model where the output is obtained out of the input via the application of successive linear combinations and nonlinear functions that are organised in layers. Symbolically, this is represented as a neural diagram.

We now evaluate a neural network using the Caret package.

#### 3.1.4.2 Model Training and Testing

```
NeuralNetModel <- train(Default ~ ., data = train_data_RERL, method = "nnet",  
                        trControl = fitControl)  
options(scipen=999)
```

```
summary(NeuralNetModel)
```

```
## a 16-1-1 network with 19 weights  
## options were - entropy fitting  
##      b->h1   i1->h1   i2->h1   i3->h1   i4->h1   i5->h1   i6->h1   i7->h1  
## 1772.42 -4731.80    -4.71    -3.37     4.43     0.90   563.61   554.21  
##      i8->h1   i9->h1  i10->h1  i11->h1  i12->h1  i13->h1  i14->h1  i15->h1  
##   218.77   559.02   574.61   517.98    -3.90    25.68    79.70  -224.61  
##      i16->h1  
##         7.58  
##      b->o     h1->o  
##      2.82    -3.97
```

```
varImp(NeuralNetModel)
```

```
## nnet variable importance  
##  
##  
##  
##          Overall  
## Term          100.00000  
## RevLineCr.T    12.12676  
## UrbanRural.1   11.89425  
## RevLineCr.N    11.79724  
## UrbanRural.2   11.69560  
## RevLineCr.Y    10.92982  
## Recession.1     4.72863  
## RevLineCr.0     4.60519  
## GuaranteedPortion 1.66546  
## RealEstate.1    0.52364  
## Recession.2     0.14115  
## NoEmp           0.08045  
## CreateJob       0.07455  
## DisbursementGross 0.06328  
## NewExist.2      0.05218  
## RetainedJob     0.00000
```

Predictions:

```
prediction.train.NN = predict(NeuralNetModel, newdata = train_data_RERL)  
prediction.test.NN = predict(NeuralNetModel, newdata = test_data_RERL)
```

### 3.1.4.3 Neural Network Performance on the Training Data

```
confusionMatrix(prediction.train.NN, train_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF    2511  789
##   PIF         340 5674
##
##           Accuracy : 0.8788
##           95% CI : (0.872, 0.8853)
##   No Information Rate : 0.6939
##   P-Value [Acc > NIR] : < 0.00000000000000022
##
##           Kappa : 0.7267
##
##   McNemar's Test P-Value : < 0.00000000000000022
##
##           Sensitivity : 0.8807
##           Specificity : 0.8779
##   Pos Pred Value : 0.7609
##   Neg Pred Value : 0.9435
##   Prevalence : 0.3061
##   Detection Rate : 0.2696
##   Detection Prevalence : 0.3543
##   Balanced Accuracy : 0.8793
##
##   'Positive' Class : CHGOFF
##
```

### 3.1.4.4 Neural Network Performance on the Testing Data

```
confusionMatrix(prediction.test.NN, test_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF     803  291
##   PIF        107 1904
##
##           Accuracy : 0.8718
##           95% CI : (0.8595, 0.8834)
##   No Information Rate : 0.7069
##   P-Value [Acc > NIR] : < 0.00000000000000022
##
##           Kappa : 0.7079
##
##   McNemar's Test P-Value : < 0.00000000000000022
##
##           Sensitivity : 0.8824
##           Specificity : 0.8674
##   Pos Pred Value : 0.7340
```

```
##          Neg Pred Value : 0.9468
##          Prevalence : 0.2931
##          Detection Rate : 0.2586
##    Detection Prevalence : 0.3523
##          Balanced Accuracy : 0.8749
##
##          'Positive' Class : CHGOFF
##
```

The sensitivity, specificity and accuracy of this neural network perform slightly worse than that of the random forest in terms of metrics, but beats it in terms of reliability (i.e. in terms of the divergence in performance on the training and testing data). Indeed, the sensitivity achieved lays around 88% in both performances and the specificity mounts to 87.79% (training data) and 86.74% (testing data). The summary showcases that the accuracy of the model with respect to correct predictions is around 87%, which is lower than the random forest method. Overall, this neural network performs relatively well. One should over note that it was voluntarily left unparametrised as the ultimate goal was to build a stacked model at a later stage. Hence, one likely might have reached even better results by optimising the hyperparameters of the model.

### 3.1.5 Extreme Gradient Boosting

#### 3.1.5.1 Model Description

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. Models are fit using any arbitrary differentiable loss function and gradient descent optimisation algorithm. This gives the technique its name, gradient boosting, as the loss gradient is minimised as the model is fit, much like a neural network. Hence, it builds the model in a stage-wise fashion.

We now evaluate an extreme grading boosting model using the 'xgbTree' method in the Caret package.

#### 3.1.5.2 Model Training and Testing

```
XGBTree <- train(Default ~ ., data = train_data_RERL, method = "xgbTree",  
                 tuneLength = 3, trControl = trainControl(method = "cv"))  
options(scipen=999)
```

```
summary(XGBTree)
```

##	Length	Class	Mode
## handle	1	xgb.Booster.handle	externalptr
## raw	96821	-none-	raw
## niter	1	-none-	numeric
## call	5	-none-	call
## params	8	-none-	list
## callbacks	1	-none-	list
## feature_names	16	-none-	character
## nfeatures	1	-none-	numeric
## xNames	16	-none-	character
## problemType	1	-none-	character
## tuneValue	7	data.frame	list
## obsLevels	2	-none-	character
## param	0	-none-	list

```
varImp(XGBTree)
```

```
## xgbTree variable importance  
##  
## Overall  
## Term 100.000000  
## DisbursementGross 8.900545  
## RetainedJob 7.205775  
## GuaranteedPortion 5.034845  
## NoEmp 3.709664  
## RevLineCr.0 2.775365  
## CreateJob 1.701819  
## UrbanRural.1 1.609979  
## RealEstate.1 1.017450  
## RevLineCr.N 0.322869  
## NewExist.2 0.206395  
## Recession.2 0.147493  
## RevLineCr.Y 0.084036  
## UrbanRural.2 0.052695  
## RevLineCr.T 0.007719
```

```
## Recession.1          0.000000
```

Predictions:

```
prediction.train.XGBTree = predict(XGBTree, newdata = train_data_RERL)
prediction.test.XGBTree = predict(XGBTree, newdata = test_data_RERL)
```

### 3.1.5.3 Extreme Gradient Boosting Performance on the Training Data

```
confusionMatrix(prediction.train.XGBTree, train_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction CHGOFF PIF
##    CHGOFF    2600  232
##    PIF         251 6231
##
##              Accuracy : 0.9481
##              95% CI : (0.9434, 0.9526)
##    No Information Rate : 0.6939
##    P-Value [Acc > NIR] : <0.0000000000000002
##
##              Kappa : 0.8777
##
##    Mcnemar's Test P-Value : 0.4128
##
##              Sensitivity : 0.9120
##              Specificity : 0.9641
##              Pos Pred Value : 0.9181
##              Neg Pred Value : 0.9613
##              Prevalence : 0.3061
##              Detection Rate : 0.2791
##    Detection Prevalence : 0.3041
##              Balanced Accuracy : 0.9380
##
##              'Positive' Class : CHGOFF
##
```

### 3.1.5.4 Extreme Gradient Boosting Performance on the Testing Data

```
confusionMatrix(prediction.test.XGBTree, test_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction CHGOFF PIF
##    CHGOFF    780  126
##    PIF         130 2069
##
##              Accuracy : 0.9176
##              95% CI : (0.9073, 0.927)
##    No Information Rate : 0.7069
##    P-Value [Acc > NIR] : <0.0000000000000002
##
```

```

##                Kappa : 0.8008
##
## Mcnemar's Test P-Value : 0.8513
##
##            Sensitivity : 0.8571
##            Specificity : 0.9426
##            Pos Pred Value : 0.8609
##            Neg Pred Value : 0.9409
##            Prevalence : 0.2931
##            Detection Rate : 0.2512
##            Detection Prevalence : 0.2918
##            Balanced Accuracy : 0.8999
##
##            'Positive' Class : CHGOFF
##

```

The xgbTree method demonstrates very good results. The sensitivity amounts to 91.2% in the training data and 85.71% in the testing data, a notable disparity. In terms of specificity, the model achieves a performance of 96.41% on the training data and 94.26% on the testing data. The accuracy reaches 94.81% (training data) and 91.76% (testing data), suggesting a good model with respect to the number of correct predictions. Overall, the disparity mentioned above might suggest that the model is overfitting on the training data, although to a lesser extent than the random forest introduced above.

## 3.2 Stacked Model

### 3.2.1 Model Development

#### 3.2.1.1 Model Description

Model stacking is an efficient ensemble method in which the predictions generated by multiple algorithms are used as inputs in a second-layer learning algorithm. This second-layer algorithm is trained to optimally combine the model predictions to form a new set of predictions. The benefit of stacking is that it can harness the capabilities of a range of well-performing models on a classification task and make predictions that boast better performance than any single model in the ensemble, although is not guaranteed to result in an improvement in all cases.

Looking at the performances of the five models above, one realises that they all display decently good performances in respect of sensitivity, specificity and accuracy. As we are building a model for loan default prediction and hence particularly concerned by its sensitivity, combining multiple high performing models, such as the SMV, neural network and extreme gradient boosting, might produce an even better performing ensemble model.

We now evaluate a stacked model combining several of the previously introduced models using the Caret package.

#### 3.2.1.2 Model Training and Testing

```
control = trainControl(method = "repeatedcv", number = 10, repeats = 3,
                        index = createFolds(train_data_RERL$Default, 10),
                        savePredictions = "final",
                        classProbs = TRUE)
```

Create a list of the algorithms to be used in the stacked ensemble.

```
algorithms <- c("glm", "svmLinear", "rf", "nnet", "xgbTree")
```

```
stackedModel <- caretList(Default ~ .,
                           data = train_data_RERL,
                           trControl = control,
                           tuneLength = 3,
                           methodList = algorithms)
```

```
stackedResult <- resamples(stackedModel)
summary(stackedResult)
```

```
##
## Call:
## summary.resamples(object = stackedResult)
##
## Models: glm, svmLinear, rf, nnet, xgbTree
## Number of resamples: 10
##
## Accuracy
##           Min.   1st Qu.   Median     Mean   3rd Qu.     Max. NA's
## glm       0.8132904 0.8218259 0.8284027 0.8271658 0.8331146 0.8381248    0
## svmLinear 0.8083025 0.8159204 0.8242872 0.8227639 0.8281343 0.8342680    0
## rf        0.8986043 0.9019742 0.9044380 0.9048745 0.9082966 0.9130279    0
## nnet      0.8544674 0.8586298 0.8598354 0.8598407 0.8627795 0.8640105    0
## xgbTree   0.9062388 0.9124366 0.9136944 0.9142153 0.9173552 0.9211405    0
##
```

```
## Kappa
##           Min.   1st Qu.   Median     Mean   3rd Qu.     Max. NA's
## glm       0.5279737 0.5486952 0.5682742 0.5656595 0.5825718 0.5994411 0
## svmLinear 0.4947616 0.5217281 0.5485437 0.5439874 0.5643257 0.5820966 0
## rf        0.7613108 0.7688420 0.7747792 0.7757248 0.7831053 0.7926822 0
## nnet      0.6563630 0.6721307 0.6799198 0.6781645 0.6876237 0.6929821 0
## xgbTree   0.7792827 0.7936775 0.7988387 0.7982207 0.8045065 0.8138676 0

stacked.glm <- caretStack(stackedModel,
                           method = "glm",
                           trControl = control,
                           metric = "Accuracy")

summary(stacked.glm)

##
## Call:
## NULL
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.7751  -0.3107   0.2339   0.2724   2.6629
##
## Coefficients:
##              Estimate Std. Error z value      Pr(>|z|)
## (Intercept)  3.52755     0.02462 143.299 < 0.0000000000000002 ***
## glm         -0.19512     0.22090  -0.883      0.377
## svmLinear    1.24682     0.23166   5.382    0.0000000736 ***
## rf          -2.34408     0.10066 -23.287 < 0.0000000000000002 ***
## nnet        -0.60849     0.06499  -9.363 < 0.0000000000000002 ***
## xgbTree     -4.76283     0.08876 -53.660 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 103264  on 83825  degrees of freedom
## Residual deviance:  39380  on 83820  degrees of freedom
## AIC: 39392
##
## Number of Fisher Scoring iterations: 6

Predictions:
prediction.train.stacked = predict(stacked.glm, type = "raw", newdata = train_data_RERL)
prediction.test.stacked = predict(stacked.glm, type = "raw", newdata = test_data_RERL)
```

### 3.2.1.3 Stacked Model Performance on the Training Data

```
confusionMatrix(prediction.train.stacked, train_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF    2609  164
##   PIF        242 6299
##
##           Accuracy : 0.9564
##           95% CI : (0.9521, 0.9605)
##   No Information Rate : 0.6939
##   P-Value [Acc > NIR] : < 0.00000000000000022
##
##           Kappa : 0.8966
##
##   McNemar's Test P-Value : 0.0001327
##
##           Sensitivity : 0.9151
##           Specificity : 0.9746
##   Pos Pred Value : 0.9409
##   Neg Pred Value : 0.9630
##   Prevalence : 0.3061
##   Detection Rate : 0.2801
##   Detection Prevalence : 0.2977
##   Balanced Accuracy : 0.9449
##
##   'Positive' Class : CHGOFF
##
```

### 3.2.1.4 Stacked Model Performance on the Testing Data

```
confusionMatrix(prediction.test.stacked, test_data_RERL$Default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CHGOFF PIF
##   CHGOFF    768  115
##   PIF        142 2080
##
##           Accuracy : 0.9172
##           95% CI : (0.907, 0.9267)
##   No Information Rate : 0.7069
##   P-Value [Acc > NIR] : <0.00000000000000002
##
##           Kappa : 0.7985
##
##   McNemar's Test P-Value : 0.1048
##
##           Sensitivity : 0.8440
##           Specificity : 0.9476
##   Pos Pred Value : 0.8698
```

```
##          Neg Pred Value : 0.9361
##          Prevalence : 0.2931
##          Detection Rate : 0.2473
##          Detection Prevalence : 0.2844
##          Balanced Accuracy : 0.8958
##
##          'Positive' Class : CHGOFF
##
```

## 4 Conclusion

This project has analysed the data necessary for classification of small companies in terms of SBA loans repayment default. First, the relevant characteristics, i.e. variables, of each companies operating in the real estate industry were introduced and their significance in the models were assessed to gain a better understanding of the data. It highlighted that the number of jobs created in these firms by means of these loans is generally very low, which could suggest that the loans guaranteed by the SBA are used for the survival and not for the expansion of the companies. This could suggest that the companies applying for those loans otherwise encounter difficulties tapping the capital markets and might show a certain bias in the health of the companies applying for this loan programme. Consequently, various classification methods have been explored in order to produce the best classifier possible. Five different classifiers have been investigated and showcased: logistic regression, SVM, random forest, neural network and extreme gradient boosting. Thereafter, a stacked ensemble classifier was trained. It proved to be highly successful in terms of both sensitivity and specificity. However, in our loan default classification problem, sensitivity is key as it is crucial to correctly identify companies which will not pay back their debt. In that respect, the extreme gradient boosting tree displayed the best sensitivity on the testing data, and should thus be preferred. A limitation needing further investigation in future data analysis resides in the lack of hyperparameters optimisation in the various models explored in this project. Interesting additional studies may consist of scrutinising the entire dataset, not just focusing on the real estate industry as it might shed light on valuable insights in terms of the industry influence on loan default.

## 5 References

- Toktogaraev, M. (2020). Should This Loan be Approved or Denied? Retrieved from kaggle: <https://www.kaggle.com/mirbektoktogaraev/should-this-loan-be-approved-or-denied>
- Basic evaluation measures from the confusion matrix. Retrieved from ClassEval: <https://classeval.wordpress.com/introduction/basic-evaluation-measures>
- Wooldridge, J. M. (2015). Introductory Econometrics: A Modern Approach. Michigan State University: Cengage Learning.