VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Magistro baigiamasis darbas

Neveiksnių paskolų dalinių išieškojimo reitingų prognozavimas

Forecasting Partial Recovery Rates for Non-Performing Loans

Indrė Baranauskaitė

VILNIUS 2020

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

STATISTINĖS ANALIZĖS KATEDRA

Darbo vadovė doc. dr. Rūta Levulienė _____

Darbo recenzentė dr. Jolita Bernatavičienė _____

Darbas apgintas

Registravimo Nr. _____

Neveiksnių paskolų dalinių išieškojimo reitingų prognozavimas

Santrauka

Remiantis tikrais skolų portfelio duomenimis, kuriuos suteikė viena skolų išieškojimo įmonė, daliniai išieškojimo reitingai po skolų portfelio nusipirkimo buvo modeliuoti naudojant vieno ir dviejų etapų modelius bei ansamblį. Vieno etapo modeliai - regresijos algoritmai - buvo taikyti su šiais metodais: atraminių vektorių mašina, atsitiktinų miškų algoritmu, k-artimiausiais kaimynais, dirbtiniais neuroniniais tinklais ir ekstremalaus gradiento auginimu. Dviejų etapų modeliai susideda iš klasifikavimo ir regresijos. Pirmame etape išieškojimo reitingai klasifikuojami į lygius nuliui ir kitus. Antrame etape taikomi regresijos algoritmai didesniems nei nulis reitingams. Klasifikavimui buvo naudoti du metodai: binarinė logistinė regresija ir atsitiktinių miškų algoritmas. Antrajame etape buvo naudoti anksčiau minėti regresijos algoritmai. Dar vienas taikytas modelis - beta perteklinių nulių ir vienetų regresija. Geriausias modelis vertinant pasirinktais suderinamumo rodikliais buvo vieno etapo atsitiktinių miškų algoritmas. Prognozavimo tikslumui pagerinti buvo naudotas apjungimo ansamblis įtraukiant tris geriausius modelius: atsitiktinių miškų, ekstremalaus gradiento auginimo ir dirbtinių neuroninių tinklų algoritmus. Įgyvendinus metodą paaiškėjo, jog ansamblis žymiai nepranoko geriausių vieno etapo modelių. Reikšmingi faktoriai buvo mokėjimų suma iki portfelio nusipirkimo ir susidariusios skolos dydis. Mėnesinių išieškojimo reitingų prognozavimui buvo naudotas atsitiktinių miškų ciklas, modeliuojantis ir prognozuojantis išieškojimo reitingus pamėnesiui.

Raktiniai žodžiai : neveiksni paskola, išieškojimo reitingai, mašininis mokymasis, prognozavimas

Forecasting partial recovery rates for non-performing loans

Abstract

Based on real debt portfolio data provided by debt collection company, partial recovery rates post portfolio acquisition were modelled using one-stage, two-stage models and models ensemble. One-stage algorithms included support vector machines, random forests, k-nearest neighbors, artificial neural networks and extreme gradient boosting. Two-stage models classified recovery rates into zeros and others as stage one and then applied regression to observations higher than zero. Classification methods used were binary logistic regression and random forests. For the second stage, same regression algorithms were used as mentioned before. Moreover, the beta one and zero inflated regression was also implemented, modelling one and zero with Bernoulli random variable and other values with beta regression. The best model evaluated on test data was one-stage random forests. Ensemble stacking was built out of three best performing models: random forests, extreme gradient boosting and artificial neural networks. However, ensemble outperformed one-stage random forests marginally. The most significant variables were found to be pre-acquisition collections and debt amount at loan default. Additionally, monthly forecast machine was implemented using a loop of random forests.

Key words : non-performing loans, recovery rates, machine learning, forecasting

Contents

Chapter 1

Introduction

Non-performing loans are loans that fail to repay their amount according to a plan and shortly can be referred as debts. Formal description [17] suggests a loan becomes a debt after the absence of payment exceeds 90 days. It is of a loan issuers interest to aggregate a set of debts and sell it in the non-performing loan (NPL) auction to debt collection companies. Potential buyers are distributed with debt data and are required to produce a price acceptable to both parties. The key variable of price is debt recovery rate (RR), which is a percentage achievable to recover a debt as of its balance amount. Additionally, loss given default (LGD) is one minus recovery rate. One of the main tasks of debt collection companies is to forecast recovery rates based on the data provided by seller and own expertise.

One European debt collection agency donated anonymised debt portfolio data set including account and payment level information. With this data, it was sought to produce models which could forecast recovery rates post debt portfolio acquisition. Recovery process for this portfolio was not full, so the models were applied to partial recovery rates. Moreover, since recovery rates were zero inflated, the hypothesis was raised that two-stage models separating zeros first and then modelling others with regression or zero-one inflated model should produce more accurate forecasts than one-stage models, which applied regression only. This was believed due to the fact that incomplete collection process resulted in extreme excess of zeros in recovery rates. Therefore, they should be treated differently.

Goal: Forecast partial recovery rates post portfolio purchase for different periods of recovery time using one-stage, two-stage models and ensemble.

Tasks:

1) Apply one-stage regression algorithms to the modelling of recovery rates, which includes: support vector machines, random forests, k-nearest neighbors, artificial neural networks and extreme gradient boosting.

2) Apply two-stage algorithm to model recovery rates. The first stage classifies rates into zeros and others using binary logistic regression and random forests classifier. Second stage applies regression to higher than zero recovery rates using same regression methods as in one-stage. Additionally, as a two-stage approach apply zero and one beta inflated model, which models zeros and ones with Bernoulli distribution and for others uses beta regression.

3) Extract the key variables having significance in modeling recovery rates.

4) Build ensemble of best performing models using stacking technique.

5) Build models on monthly recovery rates and produce monthly forecasts.

This Master thesis focuses not only on different modeling approaches, but also concerns predicting RR using ensemble stacking technique. Furthermore, time needed for debt recovery is also acknowledged by modeling recovery rates cumulatively achieved during partial collection. In addition, monthly recovery rates are also modeled. Building a model that predicts monthly recovery rates could be beneficial to any debt collection company as it would fasten new portfolio analysis process. Once the model is built, the only thing left would be to predict monthly recovery rates for the new data.

For implementation, computer software **R** was used, with packages noted in the Appendix 8.

The structure of the Master's thesis is the following. To begin with, provided literature overview introduces with relevant papers used to base further analysis. Then the methodology is outlined implemented in the thesis. Next with the methodology provided follows its implementations on the data of interest. After a number of methods is applied, ensemble stacking is built using the best performing models. Also, a loop of random forests is imple-

mented to model monthly recovery rates. Lastly, the thesis ends with results and conclusions of analysis done.

Literature overview

Articles about loss given default (LGD is one minus RR) experienced a breakthrough in mid-2000s. This was influenced by Basel II publication [2]. Basel II was a set of international banking regulations provided by Basel Committee on Bank Supervision [1], where financial institutions were endorsed to allow scientists to use their debt internal data to measure credit risk variables of their portfolios. Early publications included J. Dermine and C. Neto de Carvalho's article [12], where authors introduced LGD modelling with mortality rates for different periods. Different approach was suggested in [6] by R. Calabrese and M. Zenga. Here the authors modelled recovery rates as a mixed random variable and the mixture of beta kernels estimator. Later, in this work's continuation [7], written by R. Calabrese, it was proposed to use mixture of a Bernoulli and a beta random variables as a mixed random variable to model RR. Another article [20] published by M. Qi, X. Zhao, concerned implementing machine learning algorithms to model LGD. The paper included analysis of parametric methods - ordinary least squares, fractional response, inverse Gaussian regressions and inverse Gaussian regression with beta transformation as well as artificial neural networks and regression trees as non-parametric methods. As compared to other publications [12], [6], [7], where authors concentrated on developing one type of the model, in [20] it was suggested to test different types of models and compare their accuracy. It was proved that non-parametric models proposed outperformed parametric. In early 2010s, G. Loterman, I. Brown, D. Martens, C. Mues and B. Baesens presented the article [16], where authors introduced with in total twelve different linear and non-linear regression algorithms for LGD. Another publication comparing the number of diverse models by T. Bellotti and J. Crook [4] also recommended using a decision tree model, which implied a multi-step procedure. First, using binary logistic regression to classify observations into LGD equal to zero and

others, and then another binary logistic to split LGD equal to one and others. Remaining values were modelled with regression. It is worth-mentioning that modelled LGD was also partial. Other publication written by M. Oliveira Jr, F. Louzada, G. Pereira, F. Moreira and R. Calabrese in [19] analysed the multimodality and zero-one inflation of LGD. Here the authors proposed a mix of degenerate distributions to handle inflation. The multimodality observed in other cases was solved by using inflated mixture of beta regressions. One of the latest papers on RR were written in 2019. In [13] H. Ye and A. Belotti's publication the authors modelled recovery rates from debt collection agency point of view after purchasing a portfolio. Here the number of methods was applied: linear regression, linear regression with Lasso, beta regression and inflated beta regression. In addition, it was suggested to use a two-stage model: a beta mixture model combined with a logistic regression model, which was a similar proposal to that described in [4], but for the regression authors decided to use a beta mixture model in order to deal with multimodality, which eventually gave the best model performance. Another paper published by A. Bellotti, P. Gambetti, D. Brigo and F. Vrins [3] was of focus on machine learning. In total, twenty different machine learning algorithms were applied and described in this work.

A few theoretical articles were also concerned in this thesis. This included two articles on support vector machines presented by T. Trafalis and H. Ince in [22] and C. Cortes, V. Vapnik in their work [9]. Moreover, beta inflated model was effectively presented by R. Ospina and S. Ferrari in [18]. On artificial neural networks visualisation in software R, article by M. Beck in [5] provided explanations on different graphic tools to view a neural network.

Chapter 2

Methodology

In the Master's thesis, two types of modelling approaches were proposed: one-stage and two-stage. One-stage models included regression only, while two-stage models consisted of classification and regression.

1) One-stage model approach. Most of before discussed literature concerned applying regression to model RR and LGD. To take one, in [3], several regression algorithms that authors used were support vector regression, random forests and k-nearest neighbors. These methods as well as artificial neural networks proposed in articles [20] and [16] were applied as a one-stage approach. Additionally, extreme gradient boosting was also applied.

2) Two-stage model approach. The two-stage model was first introduced in [4] and developed in [13]. This proposal included a multi-step procedure. The first step - classification into serviced ($RR > 0$) and unserviced cases ($RR = 0$), which are defined below this paragraph. The second stage - applications of regression algorithms to serviced ($RR > 0$) cases. In addition, the inflated beta regression model with inflation at zero and one was used. This was also considered as a two-stage approach as $RR \in \{0, 1\}$ were modelled differently than other values. Such methods are quite popular among modeling recovery rates and loss given default in [6], [7], [19] and [13].

2.0.1 Definition. Debts having their collection process successfully started by the collection

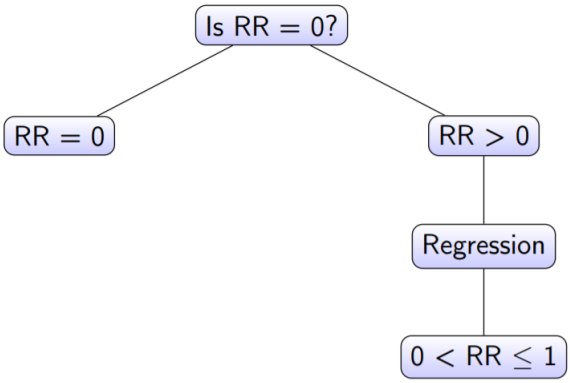

Figure 2.1: Decision tree model illustrating the two-stage approach.

company after acquisition of the portfolio are called **serviced** and their $RR > 0$. Similarly, debts with pending start or failed collection process are called **unserviced** and their $RR = 0$.

For two-stage modelling, except the beta zero and one inflated method, the second stage lead to modeling $RR \in (0, 1]$ with statistical tools discussed in 2.2. These algorithms were applied to model such cases. The decision tree figure 2.1 graphically displays the two-stage approach.

The chapter is split into sections about classification and regression methods. Classification methods included binary logistic regression and random forests. For regression such methods were applied: support vector machines, random forests, k-nearest neighbors, artificial neural networks and extreme gradient boosting. Additionally, the inflated beta regression was used, incorporating modelling one and zero values with Bernoulli random variable and remaining values with beta regression.

2.1 Classification

2.1.1 Binary Logistic Regression

The idea to use binary logistic approach for classification arose from papers [4] and [13]. In [4], the authors suggested applying logistic regression twice: firstly, to split LGD = 1 and others and then the remaining split into $LGD = 0$ and others. Binary logistic regression was also used in [13] to separate $RR = 1$ and others. In this Master's thesis, binary logistic

regression was used in two-stage modelling recovery rates, at stage one to classify debts into serviced ($RR > 0$) and unserviced ($RR = 0$).

The detailed theory on the binary logistic regression, otherwise known as the logit regression, is explained in V. Čekanavičius "Multivariate statistical analysis" lecture notes [10] with practical examples explained by the same author in his practice notes [11]. In this subsection the explanations on the binary logistic regression are based on before mentioned sources.

According to [10], the binary logistic regression belongs to the generalized linear models family, defined as follows:

2.1.1 Definition. A model for the mean of the dependent variable is called generalized linear model and has to satisfy the following conditions:

GLM: $g(\mu_j) = \beta_0 + \beta_1 X_{j1} + \beta_2 X_{j2} + ... + \beta_K X_{jK}, j = 1, ..., n.$
Here

- $\mu_j = \mathbb{E}Y_j$, and Y_j belongs to exponential family.
- The right-hand side of equality is called models *predictor*: $\eta_j(\boldsymbol{\beta}) = \beta_0 + \beta_1 X_{j1} + \beta_2 X_{j2} + ... + \beta_K X_{jK}$.
- Link function $g(\cdot)$ has inverse, that is $\mu_j = g^{-1}(\eta_j(\boldsymbol{\beta}))$.
- Parameters $\boldsymbol{\beta}$ are unknown.

If it is assumed that $g^T(\boldsymbol{\mu}) := (g(\mu_1), g(\mu_2), ..., g(\mu_n))$, $\boldsymbol{\mu}^T = (\mu_1, \mu_2, ..., \mu_n)$, then GLM can be written in matrix form

$$g(\boldsymbol{\mu}) = \mathbf{X}\boldsymbol{\beta};$$

$\mathbf{X}$ is called the design matrix, and $\boldsymbol{\beta}$ is the vector of parameters.

Binary logistic regression assumes the binomial distribution, which belongs to the exponential family of distributions and has the logit canonical link function:

- Link function $g(\mu) = ln\frac{\mu}{1-\mu}$.
- Model $ln\frac{\mu}{1-\mu} = \mathbf{X}\beta$.
- Mean estimate $\hat{\mu} = \frac{exp(\mathbf{X}\hat{\beta})}{1+exp(\mathbf{X}\hat{\beta})}$.

Typically, the values of Y are coded with 0 and 1. According to V. Čekanavičius practical examples [10], for logit regession to work, there are certain requirements for data that must not be violated:

1) The proportion of smaller group must be at least 20% of all observations, otherwise the binary logistic will not work.

2) If model contains many categorical regressors, then for each combination of their values there should be at least 5 observations in the data.

3) Regressors are not strongly correlated. Although multicollinearity is not a serious problem in logistic models, some authors recommend to drop some variables from the model if their standard errors $SE > 5$.

Overall, a good model should satisfy:

1) Maximum likelihood chi square test's $p < 0.05$.

2) Wald test's $p < 0.05$ for each regressor (all regressors are statistically significant).

3) At least 50% of cases when $Y = 1$ and at least 50% of cases when $Y = 0$ are correctly classified.

4) Cook's distance ≤ 1 and all DFBeta ≤ 1 for all observations.

5) Chosen pseudo-determinant coefficient ≥ 0.20.

Moreover, odds ratio $P(Y = 1)/P(Y = 0)$ for different variables should be calculated, showing how odds change when variable 1 is increased by a unity and others are held constant. Forecasts are made for $P(Y = 1)$. Typically, if the forecast is > 0.5, then it is assumed that the observation is classified into "1" group, otherwise "0" [11]. However, the threshold defining

whether observation is classified as "1" can be arbitrarily changed depending on the data. This is especially relevant to imbalanced data, where one class of observations is significantly lower as compared to other groups.

2.1.2 Random Forests

Random forests algorithm is not commonly used for classification in a two-stage RR models. This method was selected due to being able to handle imbalanced data sets, which was the case of our interest, as well as the ability to provide a reliable feature importance estimates.

Random forest algorithm is explicitly described in [14]. Random forests are an ensemble learning method for both classification and regression tasks. Some of the important definitions are as follows:

2.1.2 Definition. Decision tree methods are tree-based methods, which involve stratifying or segmenting the predictor space into a number of simple regions. The set of splitting rules used to segment the predictor space can be summarized in a tree.

2.1.3 Definition. Classification trees building includes two main steps:

1) The predictor space (the set of possible values for X_1, X_2,...,X_p) is divided into J distinct and non-overlapping regions, R_1, R_2,...,R_J.

2) For every observation that falls into the region R_j, the same prediction is made, which is simply most commonly occurring class of training observations in that region R_j .

The main goal here is to minimize the classification error rate E, which is the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k \hat{p}_{mk}$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

For **random forests**, a number of decision trees on bootstrapped training samples is built. When building decision trees, each time a split in a tree is considered, a random

sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically is chosen $m \approx \sqrt{p}$. By not considering total number of predictors each time, random forests lowers the correlation between the splits.

2.2 Regression

2.2.1 Support Vector Machines

Support vector machines (SVMs) were used in papers [16] and [3]. This method was chosen due to availability to use different kernels, which allowed more flexibility. Although the authors in [3] claimed that the SVM model did not achieve better accuracy than most other models they used, another publishing concerning this method [16] stated that SVM outperformed most other linear approaches. Therefore, SVM was applied in this thesis as well.

The theory of support vector machines is provided in [22] and [9] and explanations on this method are based on these sources.

SVM algorithm is based on statistical learning theory and can be applied to classification and regression. In classification case, optimal hyperplane is found that separates two classes. In order to find an optimal hyperplane, a norm of vector ω has to be minimized, which defines the separating hyperplane. This is equivalent to maximizing the margin between two classes. In the case of regression, the goal is to construct a hyperplane that lies "close" to as many of the data points as possible. Therefore, the objective is to choose a hyperplane with small norm while simultaneously minimizing the sum of the distances from the data points to the hyperplane.

To begin with, linearly separable case - the ϵ-insensitive support vector regression - is to be explained. In the ϵ-insensitive support vector regression, the idea is to find such function $f(x)$ that has an ϵ deviation from the actually obtained target y_i for all training data and at the same time is as flat as possible.

Suppose a linear function f of form:

$$f(x) = \omega x + b, \omega \in \chi, b \in \mathbb{R} \tag{2.1}$$

Small ω should be sought, which is called a support vector. The problem can be written as a convex optimization problem:

minimize

$$\frac{1}{2}\|\omega\|^2 \tag{2.2}$$

subject to

$$y_i - \omega x_i - b \leq \epsilon, \tag{2.3}$$

$$\omega x_i + b - y_i \leq \epsilon \tag{2.4}$$

The assumption in 2.3 and 2.4 is that such a function f exists that approximates all pairs (x_i, y_i) with ϵ precision. However, sometimes some errors should be allowed. Analogously to the "soft margin" loss function, slack variables ξ_i, ξ_i^* are introduced to cope with otherwise infeasible constraints of the optimization problem 2.2. Hence the formulation can be written as:

minimize

$$\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

subject to

$$y_i - \omega x_i - b \leq \epsilon + \xi_i$$

$$\omega x_i + b - y_i \leq \epsilon + \xi_i^*$$

$$\xi, \xi_i^* \geq 0, \ (i = 1, ..., m)$$

The constant $C > 0$ is called the *cost* parameter, which determines the trade-off between the flatness of f and the amount up to which deviations larger than ϵ are tolerated.

Figure 2.2 show soft margin loss setting for linear kernel SVM. This is called ϵ-insensitive loss function $|\ \xi\ |_\epsilon$ and is described by

$$|\ \xi\ |_\epsilon = \begin{cases} 0 & \text{if } |\ \xi\ | \leq \epsilon \\ |\ \xi\ | - \epsilon & \text{if } |\ \xi\ | \succ \epsilon \end{cases}$$

Then, the dual problem can be formulated by constructing the Lagrangian function:

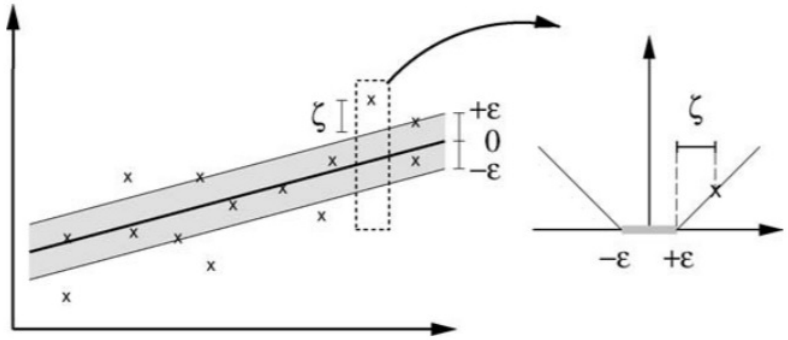

Figure 2.2: The soft margin loss setting for a linear SVM. [22]

$$
\begin{aligned}
L = \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}(\lambda_i(\epsilon_i + \xi_i - y_i + \omega\xi_i + b)) \\
- \sum_{i=1}^{l}(\lambda_i^*(y_i + \epsilon_i + \xi_i^* - \omega\xi_i - b)) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*),
\end{aligned}
\tag{2.5}
$$

here λ_i, λ_i^*, η_i, $\eta_i^* \geq 0$.

The optimal solution is the following:

$$
\begin{aligned}
\frac{\partial L}{\partial \omega} &= \omega - \sum_{i=1}^{l}(\lambda_i^* - \lambda_i) = 0 \\
\frac{\partial L}{\partial b} &= \sum_{i=1}^{l}(\lambda_i - \lambda_i^*) = 0 \\
\frac{\partial L}{\partial \xi_i} &= C - \lambda_i - \eta_i = 0 \\
\frac{\partial L}{\partial \xi_i^*} &= C - \lambda_i^* - \eta_i^* = 0
\end{aligned}
\tag{2.6}
$$

Dual problem is obtained by substituting 2.6 into 2.5. Specifically, the dual problem is as follows: maximize

$$
-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(\lambda_i - \lambda_i^*)(\lambda_j - \lambda_j^*)x_i x_j - \epsilon\sum_{i=1}^{l}(\lambda_i - \lambda_i^*) + \epsilon\sum_{i=1}^{l} y_i(\lambda_i - \lambda_i^*)
$$

Subject to

$$
\sum(\lambda_i - \lambda_i^*) = 0,
$$

here $\lambda_i, \lambda_i^* \in$(0,C).

Solving 2.5 for ω,

$$\omega^* = \sum_{i=1}^{l}(\lambda_i - \lambda_i^*)x_i$$

and by 2.1,

$$f(x) = \sum_{i=1}^{l}(\lambda_i - \lambda_i^*)x_i x + b^*.$$

Optimal value of b is computed from the complementary slackness conditions, given by:

$$\begin{aligned}
&\lambda_i(\epsilon + \xi_i - y_y + \omega^* x_i + b) = 0 \\
&\lambda_i^*(\epsilon + \xi_i^* - y_y + \omega^* x_i + b) = 0 \\
&(C - \lambda_i)\xi_i = 0 \\
&(C - \lambda_i^*)\xi_i^* = 0
\end{aligned} \tag{2.7}$$

Some conclusions could be made from the equations 2.7. First of all, only samples (x_i, y_i) with corresponding $\lambda_i = C$ lie outside the ϵ-insensitive tube around f. The set of dual variables can never be not equal to zero at the same time, λ_i, λ_l^*. If $\lambda_i \neq 0$, then $\lambda_i^* = 0$ and vice versa. Finally if $\lambda_i \in (0, C)$, then the corresponding ξ is zero. So b can be calculated:

$$\begin{aligned}
&b^* = y_i - \omega^* x_i - \epsilon \\
&\text{for} \lambda_i \in (0, C) \\
&b^* = y_i - \omega^* x_i + \epsilon \\
&\text{for} \lambda_i^* \in (0, C)
\end{aligned} \tag{2.8}$$

For the non-linear case, the problem is as follows. Firstly, input space need to be mapped into feature space and a hyperplane should be found in the feature space. The following problem is obtained:

max

$$-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(\lambda_i - \lambda_i^*)(\lambda_j - \lambda_j^*)K(x_i, x_j) - \sum_{i=1}^{l}(\lambda_i - \lambda_i^*) + \sum_{i=1}^{l} y_i(\lambda_i - \lambda_i^*)$$

subject to

$$\sum(\lambda_i - \lambda_i^*) = 0,$$

here $\lambda_i, \lambda_i^* \in (0, C)$. At the optimal solution:

$$\omega^* = \sum_{i=1}^{l} (\lambda_i - \lambda_i^*) K(x_i)$$

and

$$f(x) = \sum_{i=1}^{l} (\lambda_i - \lambda_i^*) K(x_i, x) + b$$

here $K(...)$ is a *kernel* function.

According to [9], any symmetric positive semi-definite function, which satisfies Mercer's conditions can be used as a kernel function in the SVMs context. Mercer's conditions can be written as,

$$\int \int K(x, y) g(x) g(y) dx dy > 0, \int g^2(x) dx \leq \infty$$

here

$$K(x, y) = \sum_{i=1}^{\infty} \alpha_i \psi(x) \psi(y), \alpha_i \geq 0.$$

As far as kernels are concerned, they can be chosen from a variety and the most common ones used are linear, radial, polynomial and sigmoid. A number of kernels should be tested and the best one should be chosen.

2.2.2 Random Forests

Random forests for regression was used in [3] where authors managed to obtain one of the best accuracy out of all models they applied. Explanations on random forest classification algorithm were provided in section 2.1.2. However, as compared to classification, in case of regression the main goal is not to minimize the classification error rate E, but to find such boxes R_1,...,R_J that minimize the RSS (residual sum of squares), given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{Rj})^2$$

where $\hat{y}_{Rj}$ is the mean response for the training observations within the jth box. Once the regions R_1,...,R_J have been created, the response for a given test observation is predicted by using the mean of the training observations in the region to which that test observation belongs.

2.2.3 K-Nearest Neighbours

K-nearest neighbors algorithm (KNN) is a simple method based on distance measurements and can be applied to classification and regression. In [3], authors showed that KNN was the best performing algorithm out of 20 used. Therefore, it was also included in this thesis. Explanations on KNN algorithm provided are based on statistical book [14].

For regression, given a positive integer K and a test observation x_0, the KNN algorithm first identifies the K points in the training data that are closest to x_0, represented by N_0, term "closest" meaning the shortest in calculated distance (Euclidean, Manhattan, Minkowski for numerical and Hamming for categorical variables). It then outputs the average value of these K nearest values to x_0.

The aforementioned distances between two points x_i and y_i are calculated in the following way:

Euclidean

$$\sqrt{\sum_{i=1}^{K}(x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^{K}|x_i - y_i|$$

Minkowsi

$$\left(\sum_{i=1}^{K}(|x_i - y_i|)^q\right)^{\frac{1}{q}}$$

Hamming

$$D_H = \sum_{i=1}^{K}|x_i - y_i|,$$

here if $x = y \Rightarrow D = 0$, if $x \neq y, \Rightarrow D = 1$.

The choice of K highly depends on the data. It is advised to use a large K value in order to be more precise, however, the best way to determine K value is by tuning.

2.2.4 Artificial Neural Networks

Artificial neural networks are one of the most popular modern tools of statistical modelling and implemented in many RR and LGD related papers, such as [3], [16] and [20]. The

authors there managed to achieve comparably good results in forecasting. The explanations on artificial neural networks are provided in book [15].

A technical neural network consists of simple processing units, the neurons, and directed, weighted connections between those neurons. Here, the strength of a connection (or the connecting *weight*) between two neurons i and j is referred to as $w_{i,j}$.

2.2.1 Definition. A neural network is a sorted triple (N, V, w) with two sets N, V and a function w where N is the set of neurons and V a set $\{(i,j)|i,j \in \mathbb{N}\}$ whose elements are called connections between neuron i and neuron j. The function $w : V \to \mathbb{R}$ defines the weights, where $w((i,j))$, the weight of the connection between neuron i and neuron j is shortened to $w_{i,j}$. Depending on the point of view it is either undefined or 0 for connections that do not exist in the network.

The weights can be implemented in a square weight matrix $\boldsymbol{W}$ or, optionally, in a weight vector $\boldsymbol{W}$ with the row number of the matrix indicating where the connection begins, and the column number of the matrix indicating which neuron is the target. The numeric 0 marks a non-existing connection.

Data is transferred between neurons using connections with the connecting weight being either excitatory or inhibitory.

A neuron j usually has a number of neurons with a connection to j, which transfer their output to j. For a neuron j the propagation function receives the outputs o_{i1},..., o_{in} of other neurons i_1, i_2,..., i_n (which are connected to j), and transforms them in consideration of the connecting weights $w_{i,j}$ into the network input net_j that can be further processed by the activation function. Thus, the network input is the result of the propagation function.

2.2.2 Definition. Propagation function and network input. Let $I = \{i_1, i_2, ..., i_n\}$ be the set of neurons, such that $\forall z \in \{1, ..., n\} : \exists w_{i_z,j}$. Then the network input of j, called net_j , is calculated by the propagation function f_{prop} as follows:

$$net_j = f_{prop}(o_{i1}, ..., o_{in}, w_{i_1,j}, ..., w_{i_n,j}),$$

here is the multiplication of the output of each neuron i by $w_{i,j}$, and the summation of

the results:

$$net_j = \sum_{i \in I} (o_i w_{i,j}).$$

Based on the model of nature, every neuron is always active. The reactions of the neurons to the input values depend on this activation state. The activation state indicates the extent of a neuron's activation and is often shortly referred to as activation. It's formal definition is included in the following definition of the activation function.

2.2.3 Definition. Activation state. Let j be a neuron. The activation state a_j, in short activation, is explicitly assigned to j, indicates the extent of the neuron's activity and results from the activation function.

The activation function determines the activation of a neuron dependent on network input and threshold value. At a certain time the activation a_j of a neuron j depends on the previous activation state of the neuron and the external input.

2.2.4 Definition. Activation function and activation. Let j be a neuron. The activation function is defined as

$$a_j(t) = f_{act}(net_j(t), a_j(t-1), \Theta_j).$$

It transforms the network input net_j , as well as the previous activation state $a_j(t1)$ into a new activation state $a_j(t)$, with the threshold value Θ playing an important role.

Common activation functions include binary threshold function, which can only take two values: if the input is above a certain threshold, the function changes from one value to another, but otherwise remains constant. Also, very popular is the Fermi function or logistic function:

$$\frac{1}{1 + e^{-}x},$$

which maps to the range of values $(0, 1)$, and the hyperbolic tangent, which maps to $(-1, 1)$.

An output function may be used to process the activation once again. The output function of a neuron j calculates the values which are transferred to the other neurons connected to j. More formally:

2.2.5 Definition. Output function. Let j be a neuron. The output function $f_{out} = o_j$ calculates the output value o_j of the neuron j from its activation state a_j.

Generally, both input and output functions are defined globally. Often output function is the identity, meaning that the activation a_j is directly output: if $f_{out}(a_j) = a_j$, then $o_j = a_j$.

Artificial neural networks can be of one or more layers or neurons. In practice, one layer neural network was used with 10 neurons.

2.2.5 Inflated Beta Regression

The use of beta regression to model debt RR is beneficial as beta distribution lies in $(0, 1)$ and is very suitable to model rates. Several RR and LGD related papers in past have shown implementations of beta or beta mixture regressions, such as [16] and [4]. However, beta regression does not include the end values 0 and 1, which is needed for debt RR and LGD. Inflation at 0 or 1 is common in such data. It was suggested by [6], [7], [19] and [13] to use 0 and 1 inflated beta distribution regression, where probabilities of 0 and 1 were modelled separately and interval $(0, 1)$ was modelled with the beta distribution. In this thesis data of interest, the RRs were highly inflated with zeros and had some values at 1, which suggested not only 0 inflated, but also 1 inflated model in order to incorporate both end values of RR into the model.

Further explanations on this model are based on books [21] and [18].

Since beta distribution does not include 0 and 1, it is proposed to use a mixture of a beta and Bernoulli distributions. Specifically, it is assumed that the cumulative distribution function of the random variable y is the following:

$$BEINF(y|\alpha, \gamma, \mu, \phi) = \alpha Ber(y|\gamma) + (1 - \alpha)F(y|\mu, \phi),$$

here $\text{Ber}(y|\gamma)$ represents the cumulative distribution function of a Bernoulli random variable with parameter γ and $F(y|\mu, \phi)$ is the cumulative distribution function of $\mathbf{B}(\mu, \phi)$. Here, $0 < \mu, \gamma, \alpha < 1$ and $\phi > 0$, α being the mixture parameter.

Further, using other parametrisations that are implemented in statistical software R "gamlss" package, the definition can be written as:

2.2.6 Definition. Inflated beta distribution. The probability (density) function of the

inflated beta distribution, denoted by $\mathbf{BEINF}(\mu, \sigma, \nu, \tau)$ is defined by:

$$f_Y(y|\mu,\sigma,\nu,\tau) = \begin{cases} p_0 & \text{if } y = 0 \\ (1-p_0-p_1)\frac{1}{B(\alpha,\beta)} & \text{if } 0 < y < 1 \\ p_1 & \text{if } y = 1 \end{cases}$$

for $0 \leq y \leq 1$, where $\alpha = \mu(1-\sigma^2)/\sigma^2$, $\beta = (1-\mu)(1-\sigma^2)/\sigma^2$, $p_0 = \nu(1+\nu+\tau)^{-1}$, $p_1 = \tau(1+\nu+\tau)^{-1}$, $\beta > 0$, $0 < p_0 < 1$, $0 < p_1 < 1 - p_0$. Hence $\mathbf{BEINF}(\mu, \sigma, \nu, \tau)$ has parameters $\mu = \alpha/(\alpha+\beta)$ and $\sigma = (\alpha+\beta+1)^{-\frac{1}{2}}$, $\nu = p_0/p_2$, $\tau = p_1/p_2$, where $p_2 = 1-p_0-p_1$. Hence $0 < \mu < 1$, $0 < \sigma < 1$, $\nu > 0$ and $\tau > 0$.

The expected value is given by

$$E(y) = \frac{\tau + \mu}{(1+\nu+\tau)}. \tag{2.9}$$

2.2.6 Extreme Gradient Boosting

Extreme gradient boosting (XGBoost) is an implementation of gradient boosted decision trees, which are more advanced in terms of speed and performance than regular decision trees. Simply, this method is a tree based approach, and tree based methods are one of the best performing when modelling RRs, according to recent publication [3]. The explanations on XGBoost are provided in [8] and boosting technique is defined in [14].

2.2.7 Definition. Boosting is an ensemble technique. Here new models are added sequentially to correct the errors made by existing models until no further improvements can be made.

2.2.8 Definition. Gradient boosting is a method where new models are created that predict the residuals of prior models and then added together to make the final prediction. Each tree learns from previously built trees and updates the residual errors. It uses a gradient descent algorithm (first-order iterative optimization algorithm for finding the minimum of a function) to minimize the loss when adding new models.

Gradient boosting could be split into the following steps:

- initialize the boosting algorithm with $F_0(x)$ (a function which minimizes the loss function or MSE), defined as:

$$F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma),$$

here y_i is the target variable, γ is prediction and function L is the loss function or MSE.

- compute iteratively the gradient of the loss function:

$$r_{im} = -\alpha \Big[\frac{\partial (L(y_i, F(x_i))}{\partial F(x_i)} \Big]_{F(x)=F_{m-1}(x)},$$

here α is the learning rate.

- the multiplicative factor γ_m for each terminal node is derived and the boosted model $F_m(x)$ is defined:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x),$$

here function $h_m(x)$ computes the mean of the residuals $(y - F_{m-1})$.

XGBoost algorithm has advantage over regular gradient boosting machines as the implemented algorithm allows for parallel calculations, distributed and out-of-core computations as well as cache optimizations. In XGBoost, a model is fit on the gradient of loss generated from the previous step. This method works with any differentiable loss function.

2.3 Ensemble. Stacking

According to [23], model ensembles are methods that train multiple learners and use a combination of them for forecasting. This is done for the purpose of improving the forecast accuracy and is more effective than using one model most of the times. There are many ensemble techniques available, simple ones include averaging, where forecast result is the average of multiple models predictions and weighted averaging, where the result is the average of multiple weighted models predictions. One of more advanced methods of ensemble is called *stacking*. General stacking procedure is described in figure 2.3.

Input: Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
First-level learning algorithms $\mathfrak{L}_1, \ldots, \mathfrak{L}_T$;
Second-level learning algorithm $\mathfrak{L}$.

Process:

1. **for** $t = 1, \ldots, T$: % Train a first-level learner by applying the
2. $h_t = \mathfrak{L}_t(D)$; % first-level learning algorithm $\mathfrak{L}_t$
3. **end**
4. $D' = \emptyset$; % Generate a new data set
5. **for** $i = 1, \ldots, m$:
6. **for** $t = 1, \ldots, T$:
7. $z_{it} = h_t(\boldsymbol{x}_i)$;
8. **end**
9. $D' = D' \cup ((z_{i1}, \ldots, z_{iT}), y_i)$;
10. **end**
11. $h' = \mathfrak{L}(D')$; % Train the second-level learner h' by applying % the second-level learning algorithm $\mathfrak{L}$ to the % new data set $\mathcal{D}'$.

Output: $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), \ldots, h_T(\boldsymbol{x}))$

Figure 2.3: A general stacking procedure from [23].

2.3.1 Definition. Stacking is a procedure where combined individual learners are used to train another learner. Individual learners are called first-level, while a combiner of them is called a meta-learner.

Stacking used in this Master's thesis included the following steps. First, training data was split into *ensemble-train* and *ensemble-validate* data with the chosen ratio. Then, ensemble-train data was used for training first-level models and generation of new data set for training a meta-learner. First-level and meta-learner models were different statistical models implemented.

In this Master's thesis, stacking method was applied to combine the predictability of a few best-performing models implemented.

2.4 Goodness-of-fit metrics

The model goodness-of-fit is compared using these metrics: the root mean squared error (RMSE), the mean absolute error (MAE) and the coefficient of determination (R^2). The definitions of metrics are defined below:

2.4.1 Definition. Root mean squared error.

$$RMSE = \sqrt{\sum_{i=1}^{n} \left(\frac{(\hat{y}_i - y_i)^2}{n} \right)}$$

here $\hat{y}_i$ are predicted values, y_i - true values, $i = 1, ..., n$, $n \in \mathbb{N}$ - number of observations.

2.4.2 Definition. Mean absolute error.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i|$$

here $\hat{y}_i$ are predicted values, y_i - true values, $i = 1, ..., n$, $n \in \mathbb{N}$ - number of observations.

2.4.3 Definition. Coefficient of determination.

$$R^2 = \frac{n(\sum_{i=1}^{n} \hat{y}_i y_i) - \sum_{i=1}^{n} \hat{y}_i \sum_{i=1}^{n} y_i}{\sqrt{\left[n \sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2\right] \left[n \sum_{i=1}^{n} \hat{y}_i^2 - (\sum_{i=1}^{n} \hat{y}_i)^2\right]}}$$

here $\hat{y}_i$ are predicted values, y_i - true values, $i = 1, ..., n$, $n \in \mathbb{N}$ - number of observations.

Chapter 3

Implementation

3.1 Data overview

The methodology described in Chapter 2 was implemented using real debt portfolio data. The data set was donated by anonymous debt collection agency from Europe and included a purchased portfolio information on 118,413 accounts and 178,861 post default payments. To begin with, table 5.1, provided in the Appendix 1, describes data variables provided. Additionally, other account level variables were created using payment level information pre-purchase, described in 5.2 in Appendix 2. The variable of interest was RR - recovery rate post portfolio acquisition. Since only 28 months of post-acquisition performance were available, the models were established on partial recovery and then on a monthly basis. The RR variable could be defined as follows:

$$RR_j = \sum_{i=1}^{k} \frac{Payments_i}{Balance_j},$$

here, j is the observation number, $i \in [1, k]$, k - the number of months post portfolio acquisition, $Payments_i$ - payments made after portfolio acquisition, $Balance_j$ - amount of debt (money) at the point of acquiring the portfolio for the j debt. The RR variable was calculated for each observation.

Certain procedures were done before models could be implemented. Firstly, outlier detection was applied to numeric variables "dbal" and "age". To inspect the outlying values for these variables, Tukey's method was used, which identified an outlier as a value not in

range $[-1.5QIR, 1.5IQR]$. The graphical results, provided in Appendix 3, are demonstrated in figure 5.1 for default balance and figure 5.2 in Appendix 4 for debt age. The graphs include boxplots and histograms of default balance and debt age comparing distributions pre and post outlier detection procedure. It is shown that excluding values below -1.5 QIR and above 1.5 QIR has a positive impact allowing the distributions to be more consistent. Overall, around 16000 observations were identified as outlying in default balance and around 4000 outlying in debt age.

Secondly, numerical values were then transformed to obtain values in interval $[0, 1]$, which is a standard procedure for statistical modelling. To obtain this, the following formula was applied:

$$\frac{x - min(X)}{max(X) - min(X)},$$

here $x \in X \subset \mathbb{R}$, X is the set of variable values.

Thirdly, categorical variables were coded as dummy variables, so that they were numeric and could be used as covariates in regression algorithms.

Regarding correlation between variables, the chart 5.3 in Appendix 5 represents the strength of Spearman's correlation between covariates and implies whether it is positive or negative. The significance level is $p = 0.01$. Non-significant correlations are left blank in the graph. The figure shows that the strongest correlations are between the variables representing payment information before portfolio purchase, which includes interactions between the following: "iavg", "pl12m", "colli", "RRl12m", "nl12m", "ctot", "cavg" and "LGDpre". While most of the relationships are positive, it can be observed that "LGDpre" variable has a negative correlation with before mentioned covariates.

The variable of interest is RR - recovery rate post acquisition. In figures 3.1 it is demonstrated that the distribution of RR is zero inflated. In fact, 90% of the data has $RR = 0$. Such cases were referred as "unserviced", meaning their collection process post purchase had not started yet. Other observations having $RR > 0$ were indicated as "serviced" and their collection process had started. In the second graph of 3.1 the multimodality of $RR \in (0, 1)$ can be identified, especially the modes are clear when $RR \approx 0.1$ and $RR \approx 0.8$. The reasons for modes in there could be that the soft collection process for observations around mode 0.1

Type	Number of debts	Mean	SD	Min	Max
Train	88,809	0.06	0.219	0	1
Test	29,604	0.06	0.218	0	1

Table 3.1: Data set split into training and test.

has been applied. Second mode around 0.8 could be explained by agreements with debtors to pay 80% of their debt in return to closing their accounts.

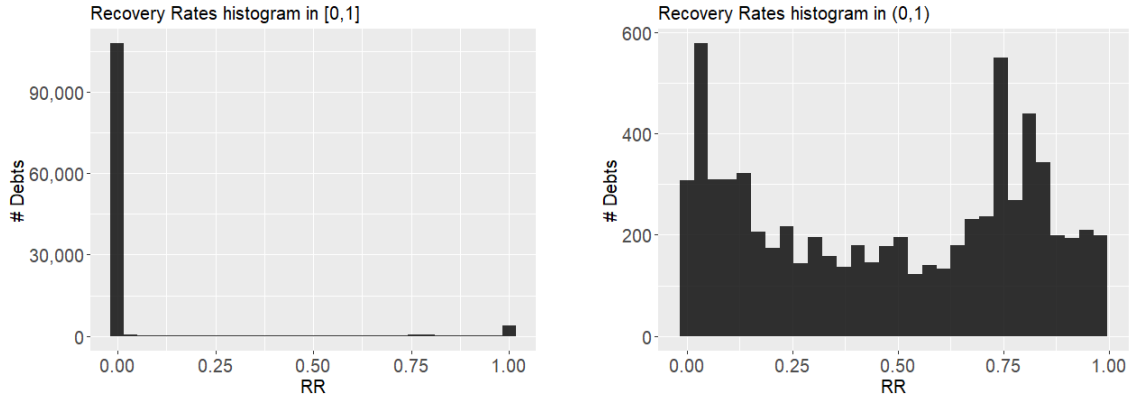

Figure 3.1: Recovery rates histograms presented in different intervals.

For models application, data was split into training and test samples with the standard ratio of 75% and 25%, showed in table 3.1. All models were applied to train data, then goodness-of-fit metrics were evaluated on test data.

In next sections of this chapter, models were implemented on the data provided. Firstly, one-stage regression models, secondly, two-stage combining classification with regression, then ensemble stacking with the best performing models. Lastly, models were built on monthly recovery rates.

3.2 One-stage models

One-stage model is the most popular way to approach the problem of modelling LGD and RR. In this Master's thesis, a number of regression algorithms was applied: support vector

regression (SVR), random forests (RF), K-nearest neighbors (KNN), artificial neural networks (ANN) and XGBoost (XGB). As a result, implemented models were evaluated in terms of their RMSE, MAE and R^2, defined in subsection 2.4.

Before mentioned algorithms did not require variable selection. Therefore, the model for such techniques was of a form:

$$RR \overset{\text{algorithm}}{\sim} dbal + age + ctot + cavg + al12m + RR.l12m + nl12m + LGDpre + pl12m + iavg + colli + dt1 + dt2 + ls1 + ls2.$$

3.2.1 Support Vector Machines

Support vector machines method for regression (SVRs) is implemented in statistical software's R package "e1071". Two different kernels were applied: radial and sigmoid.

Firstly, to implement the SVR, parameters for each kernel had to be tuned using 10-fold cross validation. Shortly, according to [14], "k-fold cross validation involves randomly dividing the set of observations into k approximately equal groups. The first fold is treated as a validation set, and the method with selected tuning parameters is fit on the remaining k-1 folds. Then, MSE_1 is calculated on the observations in the held-out fold. This procedure is repeated k times. Each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error, MSE_1, MSE_2,..., MSE_k. The k-fold CV estimate is computed by averaging these values. The final parameters are chosen the ones that achieve the lowest averaged MSE". Using 10-fold cross validation, the best performance in terms of the smallest mean squared error was reached with the parameters indicated in table 3.2. Cost parameter refers to the weight for penalizing the "soft margin", or to errors on the training points. Another parameter γ defines the influence of a single training observation.

Two SVRs with different kernels were formed with best parameters for radial and sigmoid kernels. Then unseen test data was used for prediction with these models. Table 3.3 presents that better goodness-of-fit metrics results are achieved with radial kernel SVM.

Type	Tuned hyperparameters
Radial	cost=0.12, $\gamma = 0.1$
Sigmoid	cost=1, $\gamma = 3$

Table 3.2: Tuned hyperparameters for radial and sigmoid SVMs at one-stage model.

Type	RMSE	MAE	R^2
Radial	0.22	0.08	4.60%
Sigmoid	0.25	0.09	4.33%

Table 3.3: Goodness-of-fit metrics of radial and sigmoid SVMs at a one-stage model.

3.2.2 Random Forests

Random forest algorithm (RF) has its own library implemented in R package "randomForest".

Two parameters had to be tuned in order to reach the most accurate results: number of variables taken at each split (*mtry*) and number of trees (*ntree*).

Firstly, *mtry* parameter was tuned by searching for the optimal value (with respect to out-of-bag error estimate). Out-of-bag (OOB) is the mean prediction error on each training sample x_i, using only the trees that did not have (x_i) in their bootstrap sample. Then, tuned *mtry* value was used to build a RF model with sample 1000 trees, which in addition provided results on OOB error in $ntree \in [0, 1000]$.

Tuning results are presented in figure 3.2. Here it can be observed that the best results were achieved by taking 3 variables at each split. This was not quite expected, as in 2.1.2 it was showed that recommended number of variables was $n = \sqrt{(p)}$, p - number of predictors. In this case, recommended value $mtry = \sqrt{15} \approx 3.8 \approx 4$, but the tuning revealed that the value should be 3.

The graph 3.3 shows that there is a significant drop in MSE until number of trees reaches around 100, then the error is quite stable up to 1000 trees. Extracting the smallest MSE value of *ntree*, the most optimal number of trees was 995. Overall, the number of trees was not decreased and *ntree*=1000 in the final one-stage RF model.

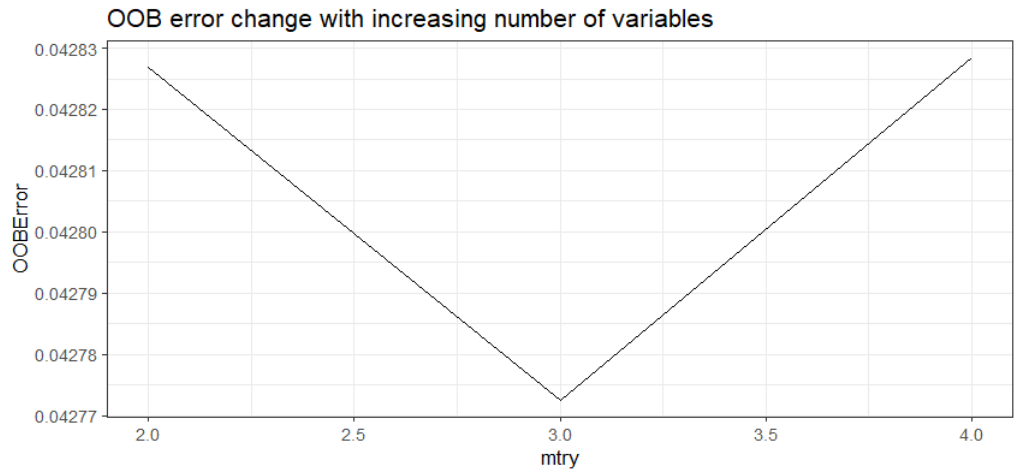

Figure 3.2: OOB error change with increasing number of variables *mtry*.

Type	**RMSE**	**MAE**	R^2
RF	0.21	0.10	9.11%

Table 3.4: Goodness-of-fit metrics results of random forests regression for the one-stage model.

The figure 3.4 demonstrates variable importance, calculated in the following way: the MSE is computed on the out-of-bag data for each tree, and then the same computed after permuting a variable. The differences between the two are averaged and normalized by the standard error. According to the graph, the most important variable in RF is "age", followed by "dbal" and "dt1", "dt2". Table 3.4 presents prediction results on test data.

3.2.3 K-Nearest Neighbors

K-nearest neighbors (KNN) algorithm is implemented R package "caret".

The number of nearest neighbors - K had to be tuned. Using a loop, 40 different KNN models were built with K $\in [1, 40]$.

The graph 3.5 displays how goodness-of-fit metrics change with regards to changing K value. RMSE drops significantly when K increases in interval $K \in [1, 10]$ and then slightly decreases until $K = 40$. In addition to this, MAE value goes down until $K = 3$, then rise

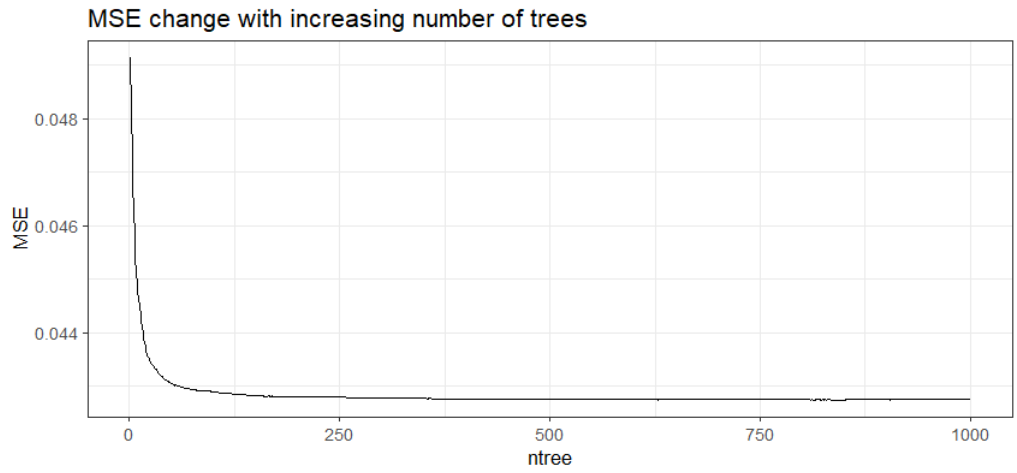

Figure 3.3: MSE change with increasing number of trees.

Type	**RMSE**	**MAE**	R^2
KNN	0.211	0.096	7.51%

Table 3.5: Goodness-of-fit metrics of KNN regession for the one-stage model.

marginally until about $K = 22$ when the value starts to decline steadily again. The increase in K value has only positive impact on R^2 value, which increases when K value grows.

Overall, the higher the K value, the better the R^2 and $RMSE$ values are achieved. For this reason, for the final KNN model K value was chosen $K = 35$, as from this value MAE started increasing again. Table 3.5 shows prediction results on test data.

3.2.4 Artificial Neural Networks

Artificial neural network (ANN) with one hidden layer was used. The method is included in R package "nnet". In addition, library "NeuralNetTools" was used for ANN visualisation. Before implemented, decay parameter had to be tuned. Decay defines the decay of weights, which both helps the optimization process and avoid over-fitting the training data. In addition, chosen number of neurons was 10.

To tune decay parameter a loop was constructed creating 10 different ANNs with changing 10 different decay values $\in [0.001, 0.01]$.

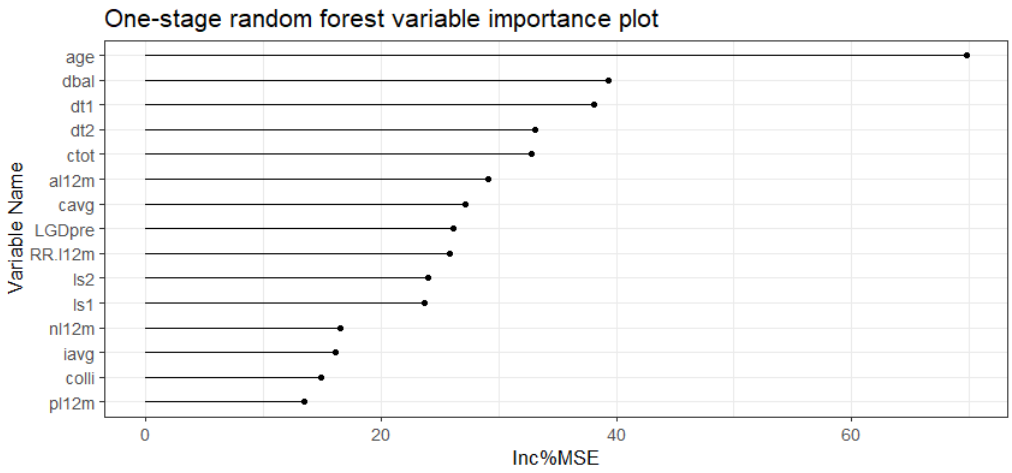

Figure 3.4: RF variable importance by %IncMSE.

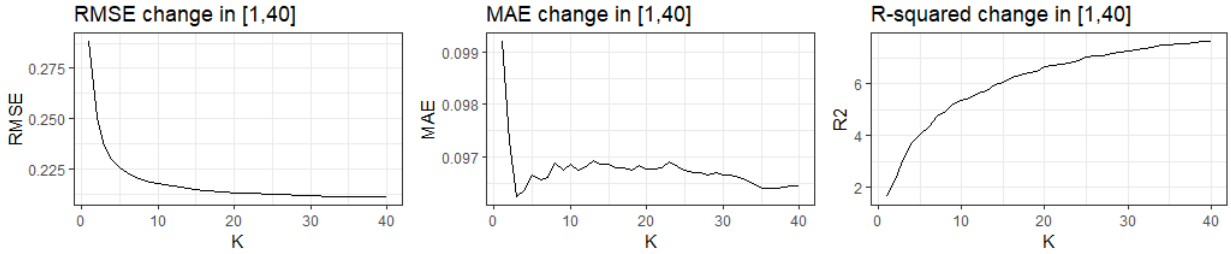

Figure 3.5: One-stage KNN goodness-of-fit metrics when $K \in [1, 40]$.

3.6 set of graphs reveals that metrics fluctuates in the interval of decays and both RMSE and R^2 seems to achieve their best results when decay is 0.009. For this reason, final ANN was built with decay value of 0.009. Table 3.6 summarises used ANNs characteristics.

The plot 3.7 shows connections between input, neurons and output. Depending on weight value a different color to line is applied. Positive weights between layers are drawn as black lines and negative weights as grey lines. Also, line thickness is in proportion to relative

Type	**Activation function**	**Number of neurons**	**Decay set**	**Tuned decay**
ANN	logistic	10	[0.001,...,0.01]	0.009

Table 3.6: ANN's used characteristics in one-stage model.

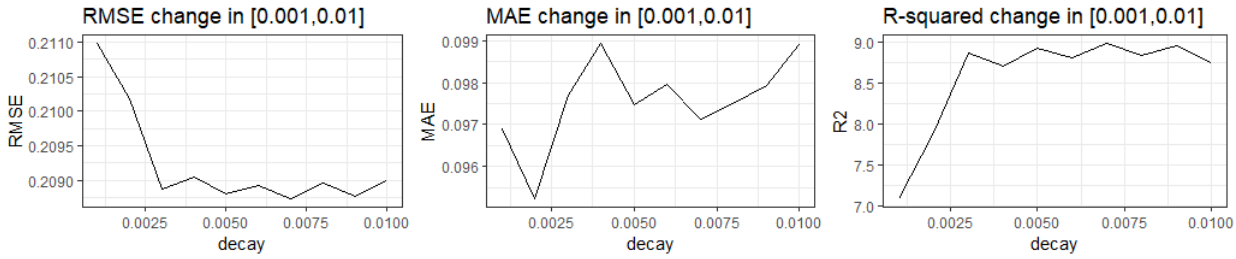

Figure 3.6: One-stage ANN goodness-of-fit metrics when decay $\in [0.001, 0.01]$

Type	**RMSE**	**MAE**	R^2
ANN	0.21	0.10	8.94%

Table 3.7: Goodness-of-fit metrics of ANN used in one-stage model.

magnitude of each weight. It can be concluded that various types of weights are assigned to all variables. It is difficult to identify variables, where connections are the strongest or the weakest. However, it can be observed that "dbal" mostly has negative weights, which can be interpreted as the higher the "dbal" value, the lower the RR.

The graph 3.8 shows variable importance of the ANN using Garson's algorithm [5], which works in the following way: "for each input node, all weights connecting an input through the hidden layer to the response variable are identified and returns a list of all weights specific to each input variable. Summed products of the connections for each input node are then scaled relative to all other inputs. A value for each input node indicates relative importance from zero to one". The figure shows that "age" is the most important variable, also pre-collection variables "LGDpre", "RR.l12m" and "nl12m" are significant in the ANN. Additionally, the table 3.7 presents model results on test data.

3.2.5 Extreme Gradient Boosting

Extreme gradient boosting (XGBoost) is implemented in R package "xgboost".

Before building the model, the following parameters presented in table 3.8 were tuned using 5-fold cross validation. With the tuned parameters, model was run incorporating all available variables. The graph 3.9 presents the most important variables - fractional

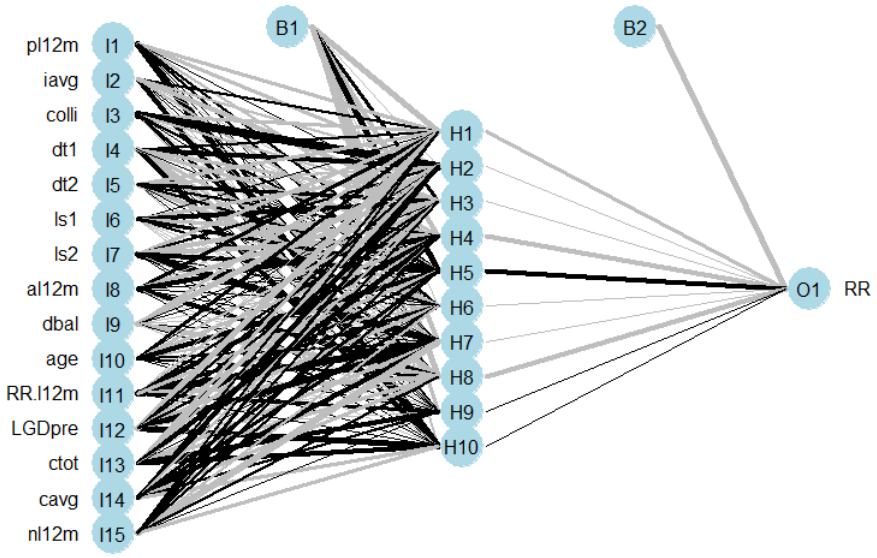

Figure 3.7: Visualisation of the ANN with 1 layer and 10 neurons in a one-stage model.

contribution of each feature to the model based on the total gain of this feature's splits. Higher percentage means a more important predictive feature. The figure shows that the most important variable is "dbal", followed by "ls1" and "ctot".

Table 3.9 presents goodness-of-fit metrics on tried method.

3.3 Two-stage models

Due to high inflation at 0, it is common to use a two-stage approach to model recovery rates. Firstly, observations are classified into $RR = 0$ and $RR > 0$. Secondly, regression algorithms are applied to model $RR > 0$. This way of modelling is used under the assumption that recovery rates equal to 0 have not been serviced yet, meaning that no actions are taken yet to start the collection process, or the collection process has been unsuccessful. Therefore, such cases need to be treated differently.

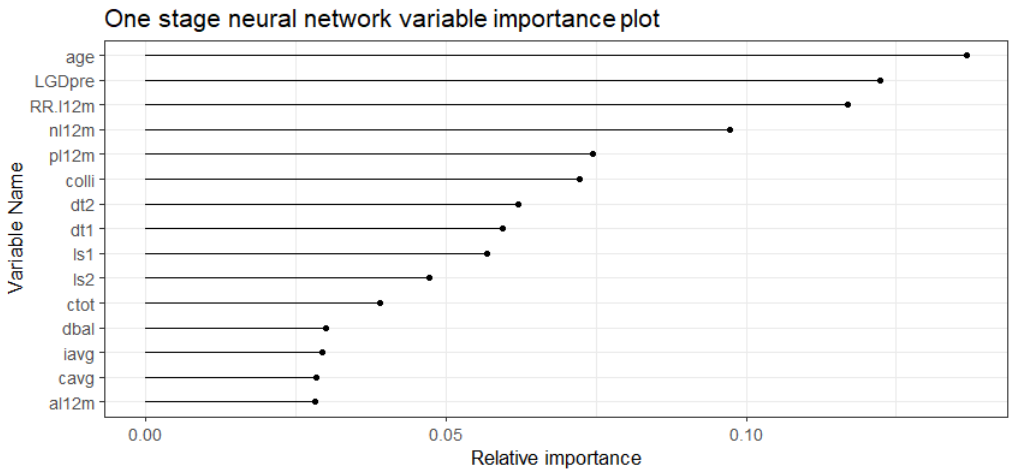

Figure 3.8: ANN variable importance plot in one-stage model.

Parameter	Definition	Tuned value
η	Controls learning rate, prevents overfit	0.4
$maxDepth$	Maximum tree depth	4
$nround$	Number of boosting iterations	25
$subsample$	Prevents overfit	0.5
$colsampleBytree$	Subsample ratio of columns	0.9

Table 3.8: Tuned parameters for XGBoost algorithms in a one-stage model.

In this Master's thesis, random forests and binary logistic classification methods were implemented as the first stage. For the second stage, regression algorithms applied in 3.2 were also applied here. Eventually, built models were evaluated in terms of their RMSE, MAE and R^2, defined in subsection 2.4.

3.3.1 Classification. Binary Logistic Regression

Binary logistic regression had the following form:

$$RR \sim dbal + age + ctot + cavg + al12m + RR.l12m + nl12m + LGDpre + pl12m + iavg + colli + dt1 + dt2 + ls1 + ls2,$$

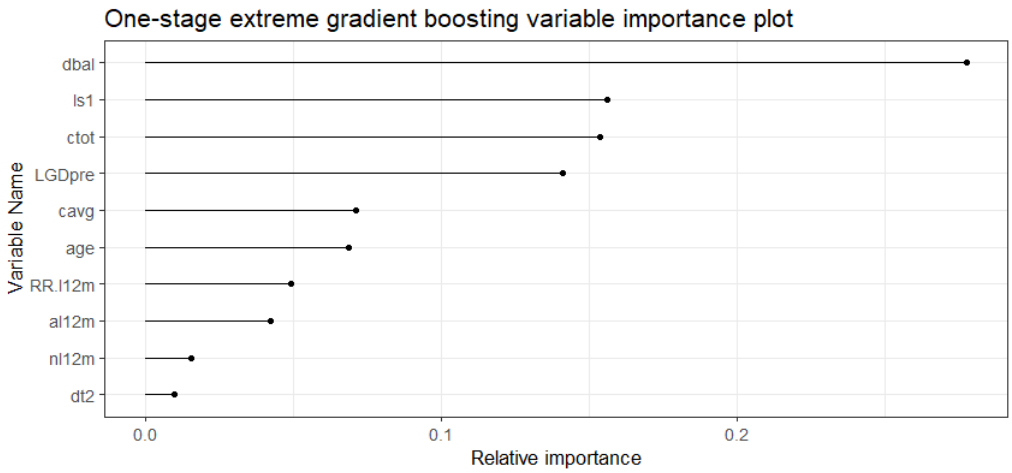

Figure 3.9: Feature importance graph of XGBoost in a one-stage model.

Type	**RMSE**	**MAE**	R^2
XGBoost	0.20	0.10	9.07%

Table 3.9: Goodness-of-fit metrics of XGBoost used in one-stage model.

here binomial family and logit link function were used. RR obtained binary values 0 or 1, with value 1 meaning recovery rates were more than zero.

The following variables were excluded as not statistically significant, when $p < 0.05$: "al12m", "ctot", "cavg", "dt1", "dt2" and "colli". The final model equation:

$$\begin{aligned}\frac{P(Y=1)}{P(Y=0)} &= -0.64 + 1.50pl12m - 0.39iavg - 0.15ls1 + 1.18ls2 + 0.26dbal \\ &\quad - 2.19age - 3.51RR.l12m - 1.52LGDpre + 4.66RR.l12m.\end{aligned} \tag{3.1}$$

Model assessment

Omnibus test

Omnibus test compares whether addition of variables perform better than model with no variables.

Firstly, a model of the following form was run:

$$RR \sim 1. \tag{3.2}$$

Analysis of Deviance Table					
Model	**Residuals**	**Df Residuals**	**Deviance**	**Df Deviance**	**Pr(>Chi)**
3.1	88799	46081			
3.2	88808	54776	-9	-8694.4	< 2.2e-16

Table 3.10: Binary logistic regression Omnibus test's results.

Omnibus test was performed in Anova Chi-squared test between 3.1 and 3.2 models. Results in table 3.10 show that the addition of variables has advantage over model with no variables, when $p < 0.05$.

Negelkerke's R-squared

Chosen pseudo R-squared statistic Negelkerke's R^2 was equal to 0.203, which just exceeded the desired threshold of 0.2. This statistic showed that the logistic model explained about 20% of fluctuations in recovery rate variable.

Cook's distance

Observations having large residuals or high leverage may distort the outcome and accuracy of a regression. Cook's distance measures the effect of deleting a given observation.

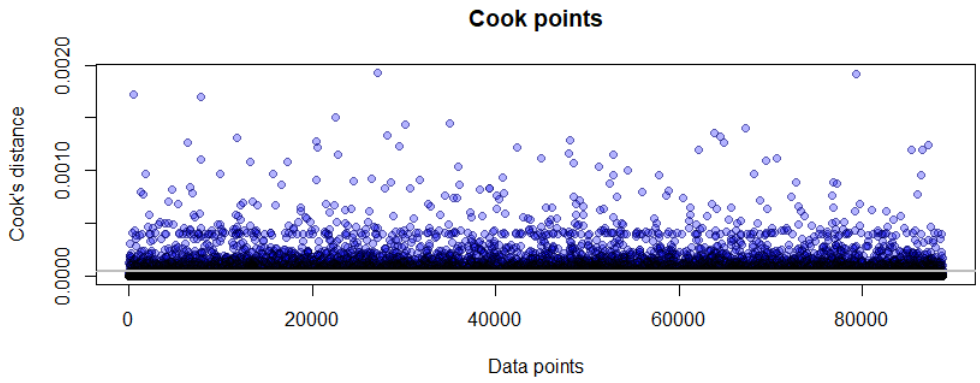

Figure 3.10: Cook's points of the binary logistic regression.

In figure 3.10 the grey line shows $\frac{4}{n}$ where n is the number of all points. Above this line the points were influential, which were 5134 points. These points altered the regression results significantly when not present in the data set.

Wald's test

Wald's test was performed in Anova test type "II". The table 5.3 in the Appendix 6 presents that all the variables are significant when $p < 0.05$.

Odd's ratio

Odds ratio for a given variable X_i can be defined as the following:

$$Odds(X_i) = e^{\beta_i},$$

here β_i is the estimated coefficient near the variable X_i. The table 5.4 in the Appendix 7 displays odds ratios for significant variables of the regression. The interpretation of odds ratio is the following: an increase of 1 in the given variable changes the log ratio by given odds ratio numeric value. Odds ratios exceeding 1 means the more variable increases, the likely the recovery rate is $RR = 1$. Such variables included: "pl12m", "ls2", "dbal" and "nl12m". On the contrary, odds ratio values lower than 1 indicates that the increase of 1 has a negative impact on recovery rate and it is more likely the recovery rate is equal to 0. Such variables were: "iavg", "ls1", "age", "RR.l12m" and "LGDpre".

Forecasting

Forecasting was done on test data. Since the data was imbalanced, to achieve the best forecasting results it was needed to change the default forecasting threshold of 0.5 to 0.15. Table 3.11 presents classification results. Although unserviced observations were classified sufficiently well with 93% correct predictions, serviced cases were classified rather poorly, with only 37% of accuracy. It was possible to achieve better classification results, so that every class would be classified at least 50% accurately, however, it was not done on purpose. By increasing classification accuracy for serviced cases, the accuracy for unserviced observations decreased significantly, which resulted in overestimation as more cases were assumed as having $RR > 0$. Therefore, higher total portfolio recovery rate was assumed.

Overall, binary logistic regression was not suitable to be used to classify recovery rates for a couple of reasons. First and foremost, the data violated the requirement that lower class should consist of at least 20% of data [10]. Second, even with classification threshold changed, still the predictions were not correct for at least 50% of each data class for test data. For these reasons, binary logistic classification was not further used in the analysis.

		Prediction	
		Unserviced	Serviced
Actual	Unserviced	93%	7%
	Serviced	63%	37%

Table 3.11: Binary logistic classification table of test data. Overall accuracy: 87%.

3.3.2 Classification. Random Forests

Random forests method has advantage over binary logistic regression that it does not have any requirements for the data and no additional tests should be applied to measure model goodness-of-fit. Therefore, even if not possible to obtain good classification results, random forests still can be used.

To build a random forest model for classification, *mtry* was tuned to be 2. Number of trees *ntree* was chosen to be 1000. Predictions were made using a majority voting rule for a certain class of each decision tree [14]. The model with $mtry = 2$ and $ntree = 1000$ was created, additionally inputting class weights to deal with imbalanced data. The optimal values of weights were chosen $RR = 0 : 0.64$ and $RR > 0 : 0.36$. With these inputs, random forests assumed that 36% of the data should be of class $RR > 0$, and 64% should be $RR = 0$. Classification results on test data are presented in table 3.12. Similarly to binary logistic regression classification, the unserviced cases were classified sufficiently well with 91% accuracy, but serviced case classification was quite poor - 43% accuracy. Overall accuracy was 87%.

All in all, random forest algorithm classified test data poorly, but slightly better than binary logistic regression. However, this method does not have any requirements to be met, so it can be used for further analysis. Eventually, good classification of unserviced cases is of greater importance than serviced, as large number of misclassified $RR = 0$ can lead to overestimation of the portfolio.

		Prediction	
		Unserviced	Serviced
Actual	Unserviced	91%	9%
	Serviced	57%	43%

Table 3.12: Random forest classification table of test data. Overall accuracy: 87%.

Type	**Tuned hyperparameters**
Radial	cost=0.2, $\gamma = 0.1$
Sigmoid	cost=0.002, $\gamma = 0.01$

Table 3.13: Tuned hyperparameters for radial and sigmoid SVMs at a two-stage model.

3.3.3 Regression

A variety of regression algorithms used in 3.2 was also implemented for the second stage regression modelling. Steps to obtain good parameter estimations and overall models were followed similarly to one-stage models.

Support Vector Machines

To implement SVMs with radial and sigmoid kernels, specific parameters had to be tuned with 10-fold cross validation, showed in table 3.13.

2 SVRs with different kernels were formed with the best parameters for radial and sigmoid kernels. Then test data was used for prediction with these models. Table 3.14 presents that better overall goodness-of-fit metrics results are achieved with radial kernel SVM.

Random Forests

Firstly, *mtry* parameter was tuned by searching for the optimal value with respect to out-of-bag error estimate. Then, tuned *mtry* value was used to build a random forests regression with sample 1000 trees, which in addition provided results on out-of-bag error in *ntree*

Type	RMSE	MAE	R^2
Radial	0.26	0.08	4.68%
Sigmoid	0.27	0.09	4.51%

Table 3.14: Goodness-of-fit metrics of radial and sigmoid SVMs at a two-stage model.

$\in [0, 1000]$.

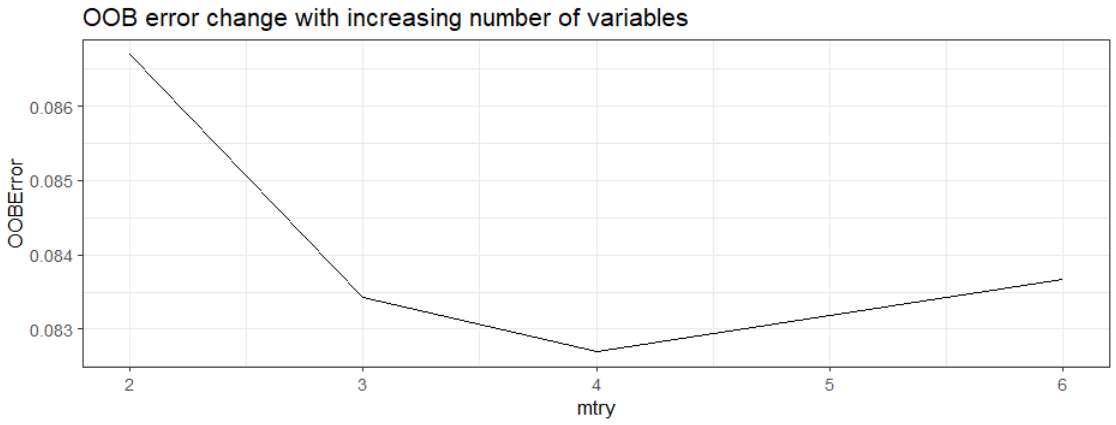

Figure 3.11: OBB error change with increasing number of variables *mtry* in two-stage model.

The figure 3.11 shows that the optimal number of variables is 4. The random forest regression was built with $ntree = 1000$ and $mtry = 4$.

In the graph 3.12 it is presented that the MSE starts to stabilize when number of trees reaches around 125. The lowest MSE is achieved when $ntree = 603$. Therefore, the model was adjusted to be of 603 decision trees.

The graph 3.13 displays variable importance in the random forest created. The top 3 most important variables are "dbal", "ls1" and "LGDpre". The overall results on predicting with test data starting with classification and then applying random forests are shown in table 3.15.

K-Nearest Neighbors

The number of nearest neighbors - K had to be tuned. Using a loop, 40 different KNN models were created with K $\in [1, 40]$. The graph 3.14 shows how goodness-of-fit metrics

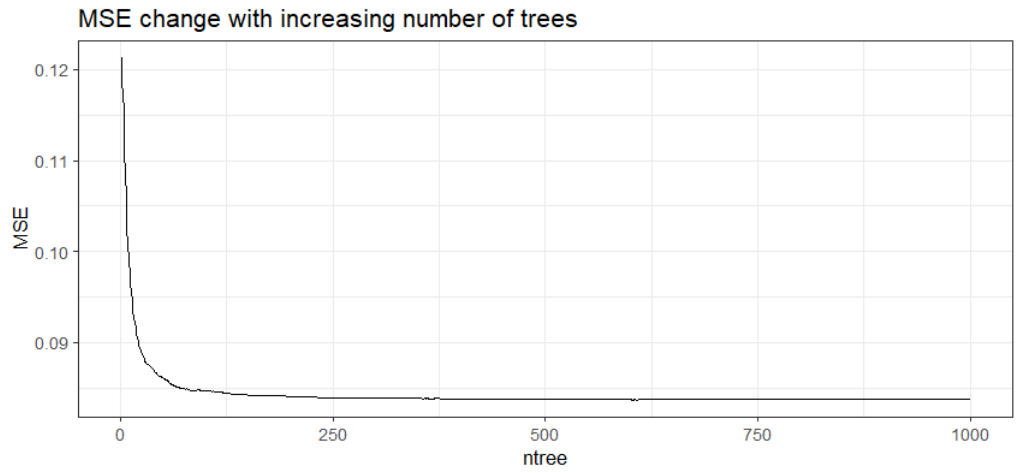

Figure 3.12: MSE change with increasing number of trees in a two-stage model.

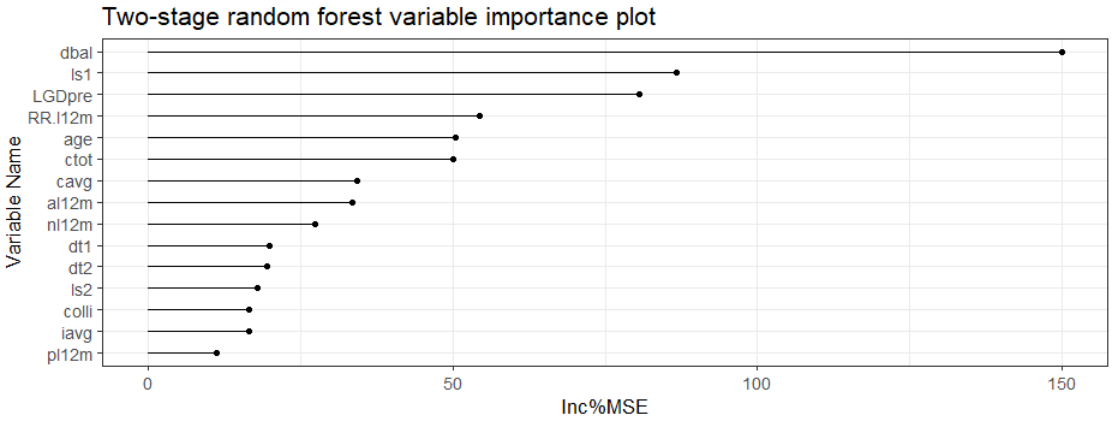

Figure 3.13: Random forests regression in two-stage model variable importance by %IncMSE.

change with regards to changing K value. While RMSE and R^2 improves over the interval, MAE reaches its lowest point at around $K = 10$, then starts to increase. K was chosen to be 30, since sufficiently good metric values were achieved at this point.

Table 3.16 presents goodness-of-fit metrics for the model built after classification task.

Artificial Neural Networks

Similarly as in one-stage model, one-layer artificial neural network with 10 neurons was used for regression. Furthermore, decay parameter responsible for optimization should be tuned in chosen range $\in [0.001, 0.01]$. To implement this, a loop of 10 different neural networks

Type	RMSE	MAE	R^2
RF	0.24	0.08	4.94%

Table 3.15: Goodness-of-fit metrics for random forests regression in two-stage model.

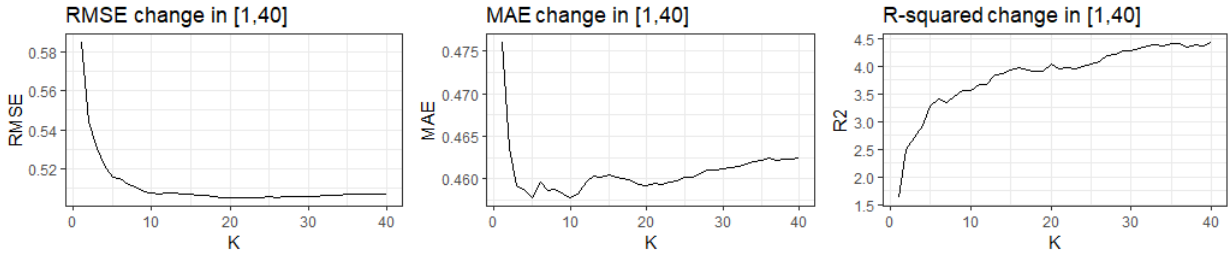

Figure 3.14: Goodness-of-fit metrics of K-nearest neighbors in two-stage model, when K $\in [1, 40]$.

was created and their goodness-of-fit metrics were compared.

The graph 3.15 demonstrates how goodness-of-fit on classified test data metrics change over the given decay interval. It can be observed that amplitudes of RMSE and MAE are quite small, meaning they do not change significantly over the interval. For this reason, the decay was chosen according to the highest R^2 value. The best R^2 value was reached when decay was 0.001, therefore this value was chosen for the final ANN model.

The plot 3.16 demonstrates neural network's connections between inputs, hidden neuron layer and output. The lines symbolize connections strength (weight) - the wider the line the stronger the connection (higher weight). Moreover, grey colour represents negative weight, while black - positive. It can be observed that the strongest negative weights are assigned to "dbal". The weakest negative weights are assigned to "dt1". It is difficult to identify the strongest positive connections, as there are many similar, but one of the strongest could be

Type	RMSE	MAE	R^2
KNN	0.25	0.08	4.83%

Table 3.16: Goodness-of-fit metrics for K-nearest neighbors regression in two-stage model.

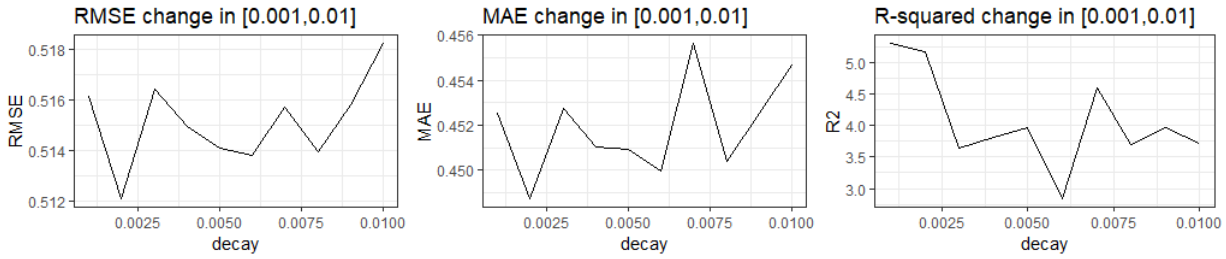

Figure 3.15: Goodness-of-fit metrics of ANN in two-stage model when decay ranges in $[0.001, 0.01]$.

Type	**RMSE**	**MAE**	R^2
ANN	0.24	0.08	4.88%

Table 3.17: Goodness-of-fit metrics for ANN regression in two-stage model.

"RR.l12m" and "nl12m".

The graph 3.17 presents variable importance plot. It can be observed that the most important variable is "dbal", then follows "RR.l12m", "pl12m" and "age". Table 3.17 presents goodness-of-fit metrics for two-stage ANN model, combining the results of classification and regression tasks.

Extreme Gradient Boosting

To implement XGBoost algorithm, specific parameters had to be tuned. The table 3.18 displays tuned parameters.

With the tuned parameters, model was run incorporating all available variables. The graph 3.18 presents the most important variables - fractional contribution of each feature to the model based on the total gain of this feature's splits. Higher percentage means a more important predictive feature. The figure shows that the most important variable is "dbal", followed by "ls1" and "RR.l12m".

Table 3.19 presents goodness-of-fit metrics on tried method, combining classification together with regression.

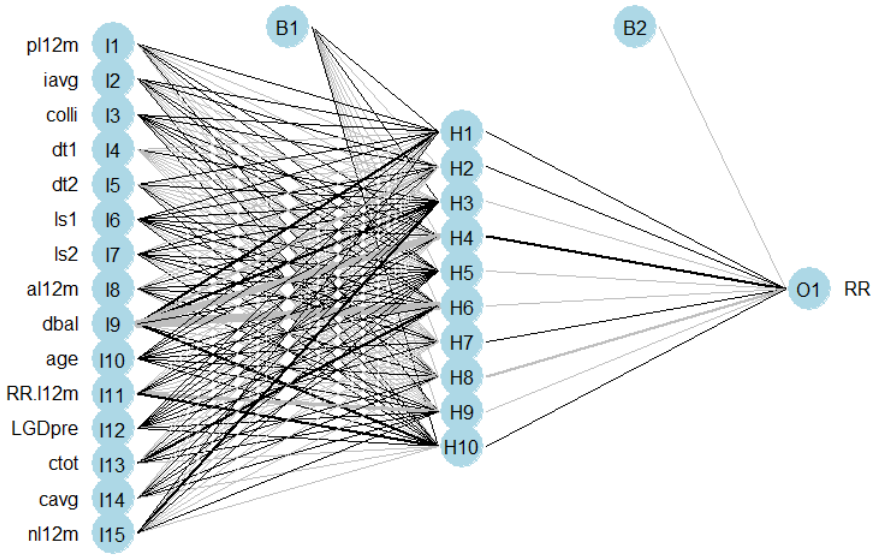

Figure 3.16: ANN plot in a two-stage model.

3.3.4 Inflated Beta Regression

0 and 1 inflation of beta distribution is implemented in R package "gamlss". This package allows modelling beta regression with inflation at 0 and 1, which is indicated as BEINF in the library. This distribution is described in 2.2.5.

First, a model without covariates was created having a form 3.2, introduced in binary logistic regression implementation section. Summary of the model for the randomised quantile residuals is the following:

- **Mean**: -0.00209956,
- **Variance**: 1.009411.

Mean is extremely close to 0 and variance is close to 1 as well, which supports BEINF distribution suitability to model recovery rates. Additionally, figure 3.19 presents normalized

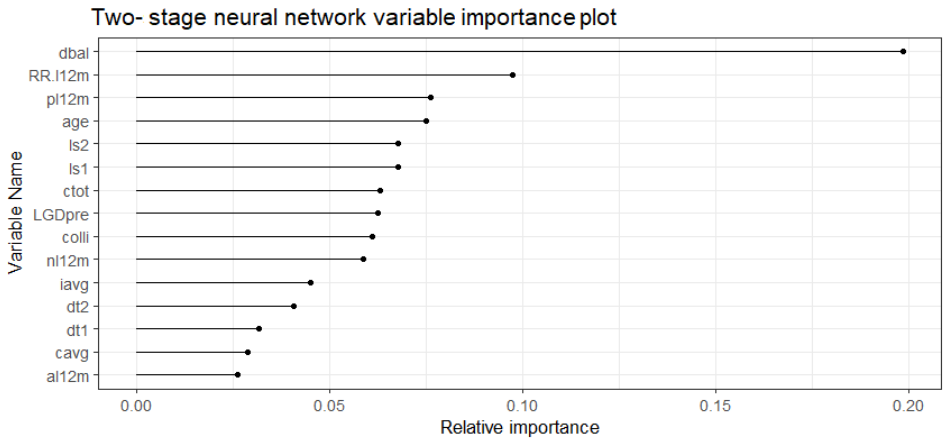

Figure 3.17: ANN variable importance plot in a two-stage model.

Parameter	**Definition**	**Tuned value**
η	Controls learning rate, prevents overfit	0.3
$maxDepth$	Maximum tree depth	5
$nround$	Number of boosting iterations	25
$subsample$	Prevents overfit	0.5
$colsampleBytree$	subsample ratio of columns	0.5

Table 3.18: Tuned parameters for XGBoost algorithms at a two-stage model.

quantile residuals in which it can be concluded that residuals are not distributed uniformly and variances are not equal (graphs *Against Fitted Values* and *Against Index*). However, *Density Estimate* graph suggests similar to normal distribution and *Normal Q-Q Plot* shows that quantiles are quite evenly distributed on the red line, with some exceptions. Overall, relying on graphs, BEINF distribution has potential to sufficiently describe recovery rate distribution, but needs further investigation.

Second, stepGAIC procedure was applied to select modelling variables for each parameter ν, σ, τ and μ, that were statistically significant. Then, a final model equations could be written. Below are represented modelled parameters with their link functions and selected variables together with outputted coefficients:

- $\log(\nu) \sim 1.38$ - 2.95pl12m + 1.89age - 0.58ls2 - 0.75dt1 + 1.90RR.l12m + 1.19LGDpre

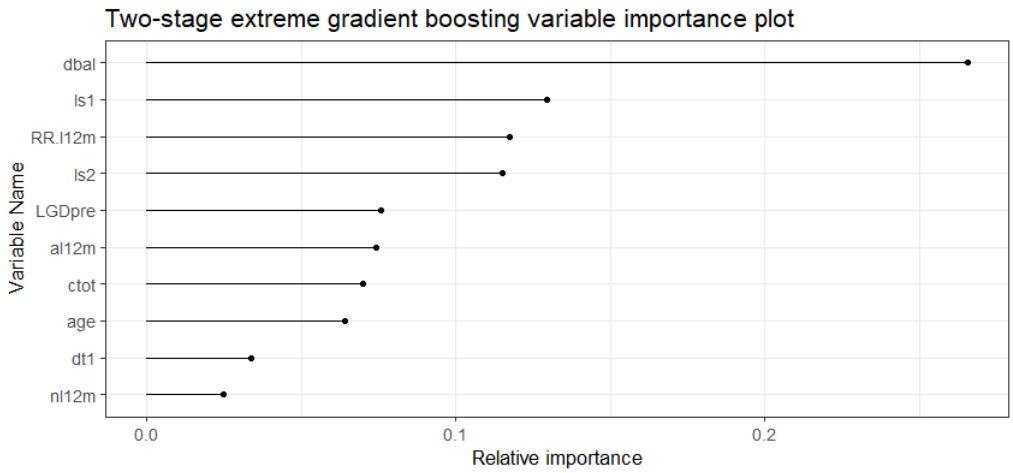

Figure 3.18: Feature importance graph of XGBoost in a two-stage model.

Type	**RMSE**	**MAE**	R^2
XGBoost	0.24	0.08	5.02%

Table 3.19: Goodness-of-fit metrics of XGBoost used in a two-stage model.

+ 1.29ls1 + 0.77iavg - 1.08dbal,

- logit(σ) $\sim$ -0.002 - 0.49RR.l12m + 0.15pl12m + 0.22ls1 + 0.19LGDpre + 0.15ls2 - 0.11nl12m,
- log(τ) $\sim$ -1.93 - 2.32nl12m + 2.17ls1 - 2.43dbal + 0.85dt1 - 0.74LGDpre - 1.26pl12m + 0.08RR.l12m,
- logit(μ) $\sim$ -2.86 - 0.20pl12m - 0.01iavg + 0.08colli + 3.11dt1 + 2.94dt2 + 0.83ls1 + 0.21ls2 + 4.12al12m - 0.10dbal - 0.56age - 0.25RR.l12m - 0.99LGDpre - 15.67ctot + 0.15nl12m.

Formed model produces the following outputs on residuals:

- **Mean**: 0.005020332,
- **Variance**: 0.9978642.

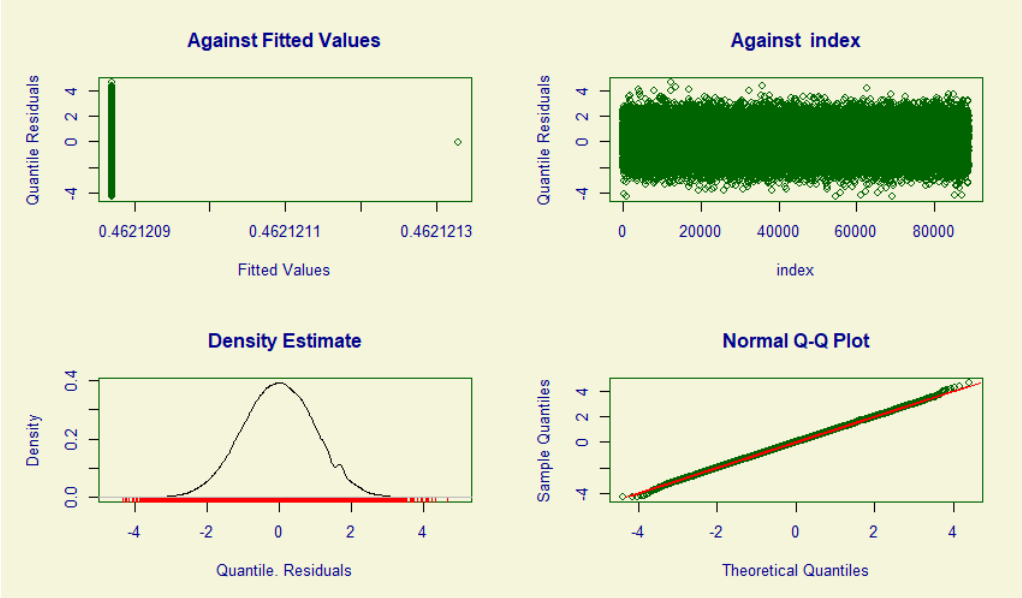

Figure 3.19: Normalized quantile residuals for a fitted BEINF model with no variables.

Again, the mean is near zero, as well as variance is close to 1. Figure 3.20 presents four graphs describing residuals. Top two graphs show that residuals are not uniformly distributed among 0 and variances are not equal. However, *Density estimate* plot indicates normal distribution and *Normal Q-Q plot* shows that quantiles are distributed among the red line, which is the desired result.

Further investigation on residuals was continued by analyzing so called *Warm plot*, which provides visual interpretations about model residuals. Ideally, residuals should be distributed around 0 and between two curved dotted lines. Graph 3.21 shows that residual values are around zero. Even though some residuals crosses the dotted lines, overall the Warm plot shows acceptable results.

Overall, although BEINF did not fit recovery rate distribution perfectly, prediction accuracy was still tested. Given the test data, model made predictions to each of new observations on parameter set (τ, μ, ν, σ). Then, the expected value was calculated using formula 2.9. Results are presented in the table 3.20.

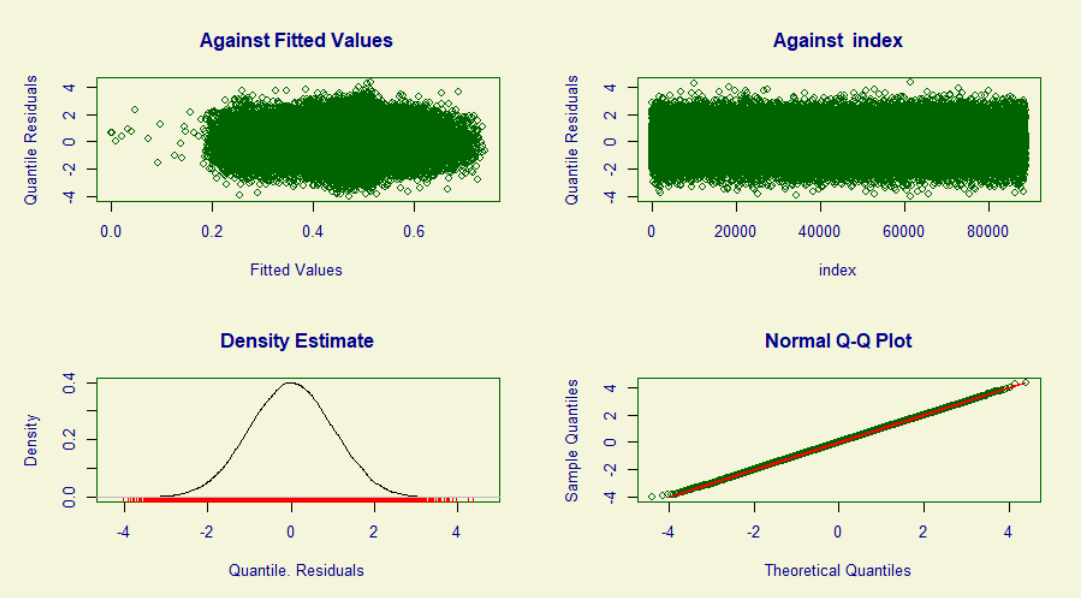

Figure 3.20: Normalized quantile residuals for BEINF model with significant variables.

Type	**RMSE**	**MAE**	R^2
BEINF	0.21	0.10	6.07%

Table 3.20: Goodness-of-fit metrics of inflated beta regression.

3.4 Models evaluation

The overall both one-stage and two-stage models goodness-of-fit characteristics can be summarised into the following table 3.21, which is split into two parts. The upper part of the table presents one-stage modeling results, and the lower part shows combined results of two-stage models. Both parts were constructed in descending order of R^2. Among one-stage methods, all of the models performed very similarly in terms of RMSE, MAE and R^2. The best performing model was RF, resulting in R^2=9.11%, and XGB was very close with R^2=9.07%. ANN produced considerably good results as well with R^2=8.94%. KNN and both radial and sigmoid kernels SVMs performed low. As far as two-stage models are concerned, it can be also observed that models performed very similarly in terms of goodness-of-fit metrics with some variations. In terms of R^2 it can be concluded that BEINF model was the best among other two-stage approaches having R^2=6.07%. The second and the third best were

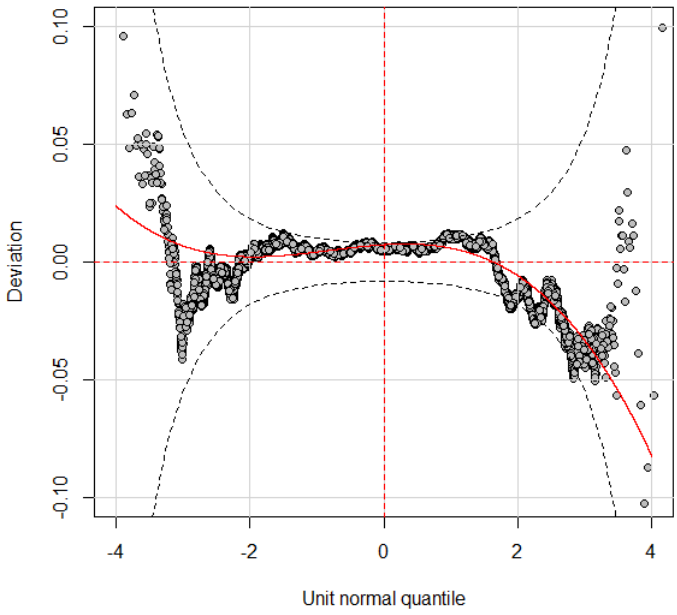

Figure 3.21: Worm plot for the inflated beta regression model.

RF&XGB and RF&RF, tree based approaches that ended up reaching similar R^2. Also, RF&ANN was not far from tree based methods with R^2=4.88%. Once again, KNNs and both SVMs performed worse than others.

All in all, one-stage models outperformed two-stage approaches, especially in terms of coefficient of determination - R^2. The best model was one-stage RF. Close in performance goodness were XGB and ANNs.

3.5 Ensemble. Stacking

To implement ensemble stacking method, training data was randomly split in half to obtain two sets: *ensemble-train* and *ensemble-validate*, which is shown in table 3.22. Ensemble-train set was used for training first-level learners: random forests, artificial neural networks and XGBoost. Then, ensemble-validate set was used for predictions on first-level algorithms. These predictions were used to train a meta-learner.

Firstly, first-level learners were built: random forests, artificial neural networks and ex-

Type	**RMSE**	**MAE**	R^2
RF	0.21	0.10	9.11%
XGB	0.20	0.10	9.07%
ANN	0.21	0.10	8.94%
KNN	0.21	0.10	7.51%
SVM-r	0.22	0.08	4.60%
SVM-s	0.25	0.09	4.33%
BEINF	0.21	0.10	6.07%
RF and XGB	0.24	0.08	5.02%
RF and RF	0.24	0.08	4.94%
RF and ANN	0.25	0.08	4.88%
RF and KNN	0.24	0.08	4.83%
RF and SVM-r	0.26	0.08	4.68%
RF and SVM-s	0.27	0.09	4.51%

Table 3.21: Overall goodness-of-fit metrics.

Type	**Number of debts**	**Mean**	**SD**	**Min**	**Max**
Ensemble-train	44,404	0.06	0.219	0	1
Ensemble-validate	44,405	0.06	0.218	0	1

Table 3.22: Training data split into $ensemble-train$ and $ensemble-validate$ for stacking.

Meta-learner	**RMSE**	**MAE**	R^2
XGB	0.21	0.10	9.30%
ANN	0.21	0.10	9.30%
RF	0.22	0.10	4.83%

Table 3.23: Goodness-of-fit metrics results with different meta-learners.

treme gradient boosting. These methods were chosen due to good performance in one-stage modelling as compared with goodness-of-fit characteristics. Corresponding parameters were newly selected using the same techniques as previously.

Secondly, first-level models were used to predict $ensemble-validate$ data set RRs. Then, obtained predicted values were gathered into one data set and inputted into $meta-learner$. Here the predicted values were treated as predictors. In order to be able to predict on the meta-learner, test data had to be used to predict values from these 3 first-level models. Table 3.23 presents results on different meta-learners. It is showed that the best meta-learners are XGB and ANN, resulting in the lowest MAE, RMSE and both reaching the highest R^2=9.3%. However, as compared to RF used in one-stage approach, where R^2=9.11%, the improvement on one-stage RF of ensemble stacking was minor, improving R^2 by only 0.19%.

3.6 Monthly forecast

Timely forecast is not a very popular concern among RR or LGD related publications. In [12], authors built models for different periods - 12, 24, 36, 48 months. In this Master's thesis, it was sought to build models even on shorter period of time - on a monthly basis. Monthly forecast was implemented using one-stage model RF:

1) One-stage RF was the best model in terms of goodness-of-fit metrics and outperformed other one-stage and two-stage approaches.

2) Even though ensemble stacking produced slightly better results than one-stage RF, ensemble stacking required a number of additional time-consuming computations. Given

Type	**RMSE**	**MAE**	R^2
RF monthly	0.03	0.001	0.12%

Table 3.24: Average goodness-of-fit metrics for RF regression in a monthly model.

that the stacking did not outperform one-stage RF significantly, it was not used as the best method for building monthly models.

The algorithm to build monthly models was the following: a loop of random forests with $i \in [1, n]$, n=28 iterations corresponding to the number of months to predict was run. In the same loop, predictions were made on test data and stored in a list. The procedure can be written as:

1. Run model: $RR_i \overset{\text{RF}}{\sim} dbal + age + ctot + cavg + al12m + RR.l12m + nl12m + LGDpre + pl12m + iavg + colli + dt1 + dt2 + ls1 + ls2$;
2. Predict: $Pr_i \overset{\text{predict}}{\sim} test\_data$;
3. Store predictions: list(Pr_1,...,Pr_n),

here $i \in [1, 28]$.

Average goodness-of-fit metrics are represented in table 3.24. The table presents that RMSE and MAE are quite small and the R^2 is also very low which indicates inappropriate model. The predicted values of test data are plotted as a curve together with the actual monthly RR curve in figure 3.22. Visually, the difference between actual and predicted curve is minor and the actual trend is maintained in the forecast. The explanation could be the following: even though debt-by-debt predictions were poor, the total portfolio recovery rate levels could be predicted sufficiently well.

Figure 3.22: Predicted monthly recovery rates versus actual.

Chapter 4

Conclusions

Overall, certain conclusions can be drawn from the Master's thesis. Firstly, concerning the one-stage approach, some differences in goodness-of-fit metrics RMSE and MAE between the models were observed. The differences in R^2 was indicated as well, especially between the best and the worst methods. The best model was random forests, achieving RMSE of 0.21, MAE 0.10 and $R^2 = 9.11\%$. Regarding the two-stage approach, similarly, some differences were indicated in RMSE and MAE and R^2. The best two-stage model was beta zero and one inflated model with RMSE equal to 0.21, MAE 0.10 and $R^2 = 6.07\%$. It can be concluded that one-stage models outperform two-stage approaches and separation of zeros was not required. Secondly, it is observed that the best two performing models as compared to goodness-of-fit metrics were tree-based. Thirdly, significant variables identified by the best models were balance at default, age of debt, legal status, debtor type and total pre-acquisition collections. Then, it was showed that ensemble stacking technique surpassed other one-stage models marginally. Lastly, monthly forecasting model was implemented with a loop of random forests.

Chapter 5

Bibliography

[1] Basel II,
`https://www.investopedia.com/terms/b/baselii.asp`.

[2] Basel Committee on Banking Supervision. International Convergence of Capital Measurement and Capital Standards A Revised Framework. *Bank for International Settlements*. 2005

[3] Bellotti, A., Gambetti, P., Brigo, D., Vrins, F. Forecasting recovery rates on non-performing loans with machine learning *Conference: Credit Scoring and Credit Control XVI*. Edinburgh, 2019.

[4] Bellotti, T., Crook, J. Loss given default models incorporating macroeconomic variables for credit cards. International Journal of Forecasting. **28** (2012), 171–182.

[5] Beck, M. *NeuralNetTools: Visualization and Analysis Tools for Neural Networks*. Journal of Statistical Software. **85(11)** (2018).

[6] Calabrese, R., Zenga, M. Bank loan recovery rates: Measuring and nonparametric density estimation. *Journal of Banking and Finance*, **34(5)** (2010), 903-911.

[7] Calabrese, R. Predicting bank loan recovery rates with a mixed continuous-discrete model. *Applied Stochastic Models in Business and Industry*, **30** (2012), 99-114.

[8] Chen, T., Guestrin, C. *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, 2016, Association for Computing Machinery, 785–794.

[9] Cortes, C., Vapnik, V. Support Vector Networks. Machine Learning **20** (1995), 273-297.

[10] Čekanavičius, V. *MULTIVARIATE STATISTICAL ANALYSIS. Lecture notes*. Vilnius 2019, 38-40.

[11] Čekanavičius, V. *5 SPSS GLM E 2018*. Vilnius 2018, 1-3.

[12] Dermine, J., Neto de Carvalho, C. Bank loan losses-given-default: A case study. *Journal of Banking and Finance*, **30** (2006), 1219–1243.

[13] Ye, H., Bellotti, A. Modelling Recovery Rates for Non-Performing Loans. *Risks*, **7(1)** (2019), 19.

[14] James, G., Witten, D., Hastie, T., Tibshirani, R. *An introduction to statistical learning with applications in R*. 2005, Springer, 39-42, 317-321.

[15] Kriesel, D. *A Brief Introduction to Neural Networks*. 2007, e-book available at `http://www.dkriesel.com`, 33-38.

[16] Loterman, G., Brown, I., Martens, D., Mues, C., Baesens, B. Benchmarking regression algorithms for loss given default modeling. International Journal of Forecasting. **28** (2012), 161–170.

[17] Non-performing loan, `https://www.investopedia.com/terms/l/loan.asp`.

[18] Ospina, R., Ferrari, S. *Inflated beta distributions*. Statistical papers. **51(1)** (2007).

[19] Oliveira Jr, M., Louzada, F., Pereira, G., Moreira, F., Calabrese, R. Inflated mixture models: Applications to multimodality in loss given default. SSRN Electronic Journal. **8** (2015).

[20] Qi, M., Zhao, X. Comparison of modeling methods for Loss Given Default. Journal of Banking and Finance. **35** (2011), 2842–2855.

[21] Rigby, B., Stasinopoulos, M. *A flexible regression approach using GAMLSS in R.* 2010, e-book available at `http://www.gamlss.com/wp-content/uploads/2013/01/book-2010-Athens1.pdf`, 215-216.

[22] Trafalis, T., Ince, H. Support vector machine for regression and applications to financial forecasting. Conference paper 2000.

[23] Zhou, Z. *Ensemble methods. Foundations and algorithms.* 2012, CRC Press, Taylor Francis Group, LLC, 83-86.

Appendix 1

Name	Type	Definition
Account level		
acc.id	factor	unique debt ID
dbal	numeric	debt amount at default
abal	numeric	debt amount at purchase
age	numeric	months from default to acquisition
dt	factor	debtor type: "1", "2", "3"
ls	factor	legal.status: "1", "2", "3"
Payment level		
acc.id	factor	unique debt ID
pmt.amt	numeric	amount paid
pmt.dt	date	date when the payment was made

Table 5.1: Debt data given variables.

Appendix 2

Name	Type	Definition
RR	numeric	ratio of collections post purchase with purchased debt amount
ctot	numeric	paid total until purchase
cavg	numeric	average payment total
al12m	numeric	average payment in last 12 months
RR.l12m	numeric	recovery rate l12m
LGDpre	numeric	loss given default until purchase
pl12m	binary	"1" - paid last 12M, "0" - opposite
nl12m	binary	number of payments last 12M until purchase.
iavg	binary	"1" - average last 12M payment increased, "0" - opposite
colli	binary	"1" - sum of payments last 12M higher than before, "0" - opposite

Table 5.2: Debt data created variables.

Appendix 3

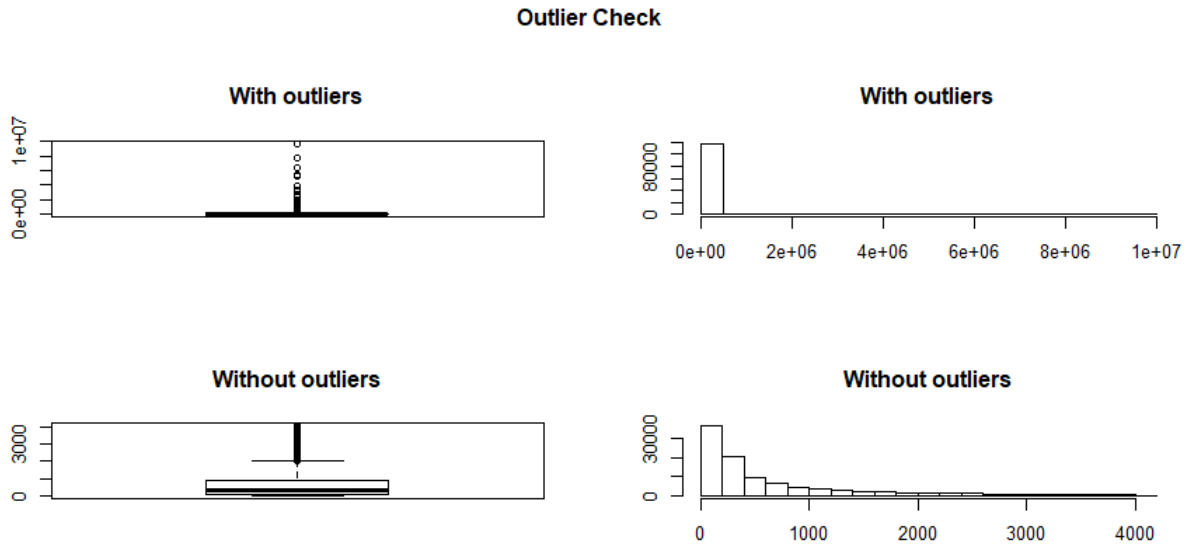

Figure 5.1: Default balance distribution with and without outliers.

Appendix 4

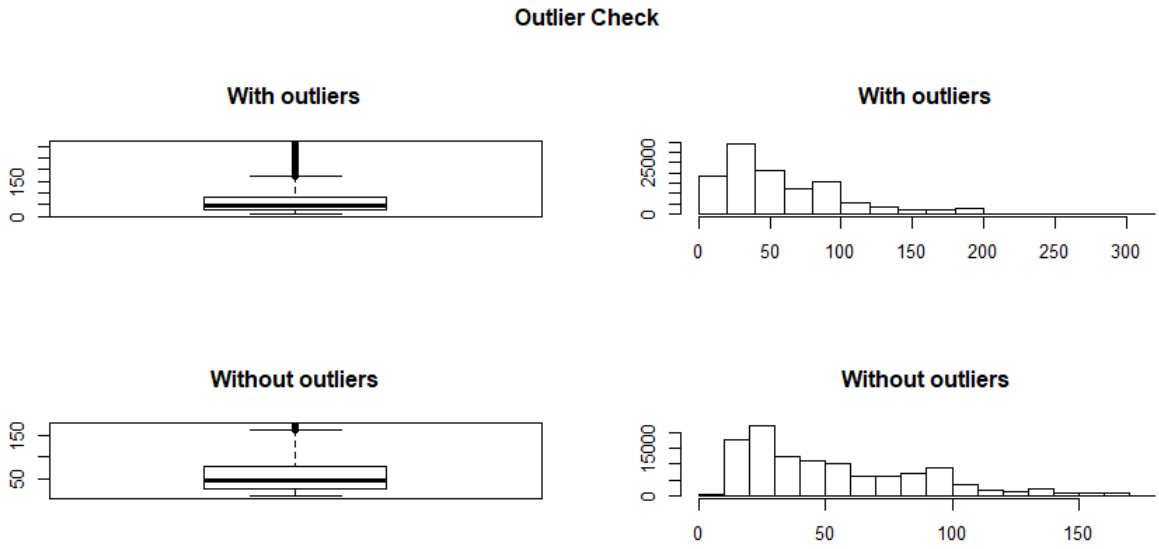

Figure 5.2: Debt Age distribution with and without outliers.

Appendix 5

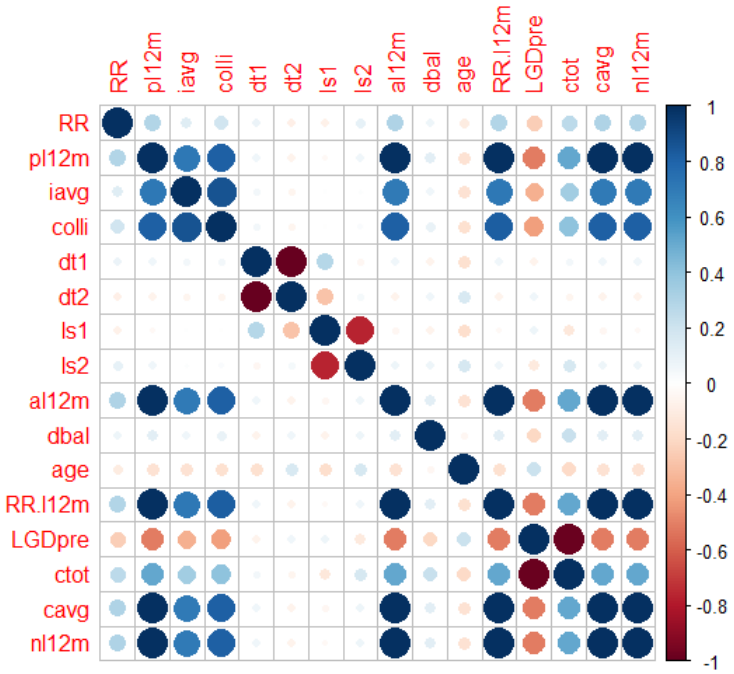

Figure 5.3: Correlation plot of the variables.

Appendix 6

Analysis of Deviance Table (Type II tests)			
Response: RR			
Variable	**Df**	**Chisq**	**Pr(>Chisq)**
pl12m	1	609.373	< 2.2e-16
iavg	1	38.138	6.591e-10
ls1	1	4.709	0.03001
ls2	1	232.917	< 2.2e-16
dbal	1	18.116	2.079e-05
age	1	753.604	< 2.2e-16
RR.l12m	1	462.024	< 2.2e-1
LGDpre	1	424.356	< 2.2e-16
nl12m	1	827.530	< 2.2e-16

Table 5.3: Logistic regression: Wald test.

Appendix 7

Variable	**Odds ratio**
pl12m	4.48
iavg	0.68
ls1	0.86
ls2	3.25
dbal	1.29
age	0.11
RR.l12m	0.03
LGDpre	0.22
nl12m	105.57

Table 5.4: Logistic regression: odds ratios for significant variables.

Appendix 8

R codes

```
# data structuring packages:
library(dplyr); library(tidyr); library(scales); library(lubridate);
# plotting packages:
library(ggplot2); library(corrplot);
# regression and classification algorithms packages:
library(caret); library(randomForest); library(mlbench);
library(e1071); library(rpart); library(stats);
library(aod); library(nnet); library(NeuralNetTools);
library(gamlss); library(gamlss.inf); library(xgboost)
# data structuring and normalization
Total_population <- Accounts%>%select(Acq.Balance,  Debt.Age, Def.Balance,
Debtor.Type, Legal.Status, Coll.Tot,Avg.Pmt.L12M, Liq.L12M, N.Pmts.L12M,
Payer.L12M, Balance.rng,RR, LGD_pre, Avg.Pmt.before.L12M, Coll.before.L12M,
Coll.L12M)%>% mutate(Debtor.Type = as.factor(Debtor.Type),
         Legal.Status = as.factor(Legal.Status),
         Avg.Pmt.increased = ifelse(Avg.Pmt.before.L12M<Avg.Pmt.L12M, 1, 0 ),
         Coll.increased = ifelse(Coll.before.L12M<Coll.L12M, 1, 0 ),
         Payer.L12M = Payer.L12M
         )%>%
  select(RR, Def.Balance, Debtor.Type, Debt.Age, Legal.Status, Coll.Tot,
  Coll.L12M, Avg.Pmt.L12M, Liq.L12M, N.Pmts.L12M, Payer.L12M, LGD_pre,
```

```
         Avg.Pmt.increased, Coll.increased)
# outlier detection
outlierKD(Total_population, Def.Balance)
outlierKD(Total_population, Debt.Age)
# correlation
corrplot(cor(Total_population,method = "spearman"),sig.level = 0.01, )
# split into test and train
smp_size <- floor(0.75 * nrow(Total_population))
train_ind <- sample(seq_len(nrow(Total_population)), size = smp_size)
train <- Total_population[train_ind, ]
test <- Total_population[-train_ind, ]
train_non_factors <- train%>%select(-RR,-Debtor.Type,-Legal.Status,-Payer.L12M,
-Avg.Pmt.increased,-Coll.increased)
train_factors <- train%>%select(RR,Debtor.Type,Legal.Status,Payer.L12M,
Avg.Pmt.increased,Coll.increased)
maxmindf <- as.data.frame(lapply(train_non_factors, normalize))
maxmindf <-bind_cols(maxmindf,train_factors)
test_non_factors <- test%>%select(-Debtor.Type,-Legal.Status,
-Payer.L12M,-Avg.Pmt.increased,-Coll.increased)
test_factors <- test%>%select(Debtor.Type,Legal.Status,
Payer.L12M,Avg.Pmt.increased,Coll.increased)
maxmindf_test <- as.data.frame(lapply(test_non_factors, normalize))
maxmindf_test <-bind_cols(maxmindf_test,test_factors)
cl_maxmindf<-maxmindf%>%mutate(RR_0_1 = ifelse(RR>0,1,0),
RR_others = ifelse(RR==0 | RR==1, 1, 0))%>%
  mutate(RR1 = ifelse(RR=="1", 1, 0),Payer.L12M  = as.factor(Payer.L12M ),
Avg.Pmt.increased = as.factor(Avg.Pmt.increased),
Coll.increased = as.factor(Coll.increased))%>%
select(-RR_others, -RR1,-RR_0_1)
cl_maxmindf_test<-maxmindf_test%>%mutate(RR_0_1 = ifelse(RR>0,1,0),
```

```
RR_others = ifelse(RR==0 | RR==1, 1, 0))%>%mutate(RR1 =
ifelse(RR=="1", 1, 0),Payer.L12M  = as.factor(Payer.L12M ),
         Avg.Pmt.increased = as.factor(Avg.Pmt.increased),
         Coll.increased = as.factor(Coll.increased))%>%
  select(-RR_others, -RR1,-RR_0_1)
cl_maxmindf.v2 <-cl_maxmindf%>%mutate(RR=ifelse(RR==1,1,RR))%>%
  mutate(RR=round(RR,2))
cl_maxmindf_test.v2 <-cl_maxmindf_test%>%mutate(RR=ifelse(RR==1,1,RR))
  mutate(RR=round(RR,2))

# One-stage
# SVM
tune.out.3=tune(svm ,RR~.,data=cl_maxmindf.v3.v2 ,kernel ="radial",
ranges =list(gamma = seq(0.1,2,length=5), cost=seq(0.12,1,length=5)))
tune.out.4=tune(svm ,RR~,data=cl_maxmindf.v3.v2 ,kernel ="sigmoid",
ranges =list(nu = seq(1,5,length=5), cost=seq(0.01,1,length=5)))
SVM.all.radial <- svm(RR ~ ., data = cl_maxmindf.v3.v2, kernel = "radial",
             cost=0.12, gamma=0.1)
SVM.all.sigmoid <- svm(RR ~ ., data = cl_maxmindf.v3.v2, kernel = "sigmoid",
             cost=1, nu=3)
pre_SVM.radial <- predict(SVM.all.radial,pre.test)%>%
as.data.frame()%>%rename(Pr=".")
pre_SVM.sigmoid <- predict(SVM.all.sigmoid,pre.test)%>%
as.data.frame()%>%rename(Pr=".")

RMSE(pre_SVM.radial$Pr, cl_maxmindf.v3.v2.test$RR)
MAE(pre_SVM.radial$Pr, cl_maxmindf.v3.v2.test$RR)
R2(pre_SVM.radial$Pr, cl_maxmindf.v3.v2.test$RR)*100
RMSE(pre_SVM.sigmoid$Pr, cl_maxmindf.v3.v2.test$RR)
MAE(pre_SVM.sigmoid$Pr, cl_maxmindf.v3.v2.test$RR)
```

```
R2(pre_SVM.sigmoid$Pr, cl_maxmindf.v3.v2.test$RR)*100

# Random forests
pre.RFAll <- cl_maxmindf.v3.v2[,-1]
res.RFAll <- cl_maxmindf.v3.v2[,1]
tune.RF.All <- tuneRF(x=pre.RFAll, y=res.RFAll, mtryStart = 2, ntreeTry = 1000,
stepFactor = 1.5,improve=1e-5,plot = TRUE,trace = TRUE)
rfAlltune <- randomForest(x=pre.RFAll, y=res.RFAll, mtry=3,
ntree = 1000, importance=TRUE)
RMSE(pr.AllRF, cl_maxmindf.v3.v2.test$RR)
MAE(pr.AllRF, cl_maxmindf.v3.v2.test$RR)
R2(pr.AllRF, cl_maxmindf.v3.v2.test$RR)*100
prRFall<-pr.AllRF%>%as.data.frame()%>%rename(RR=".")%>%
mutate(Type=paste("Predicted"))

# K-nearest neigbors
#tune the K
pre.RFAll <- cl_maxmindf.v3.v2[,-1]
res.RFAll <- cl_maxmindf.v3.v2[,1]
pre.test <- cl_maxmindf.v3.v2.test[,-1]
knn.list <- list()
n<-40
for (i in 1:n) {
  fit_knn_All = knnreg(x = pre.RFAll, y = res.RFAll, k=i )
  knn.list[[i]]<-fit_knn_All
  rm(fit_knn_All)}
knn.list.pr <- list()
for (i in 1:n) {
  pr.Allknn = predict(knn.list[[i]],newdata = pre.test)
  knn.list.pr[[i]]<-pr.Allknn
```

```
  rm(pr.Allknn)}
knn.list.r2 <- list()
for (i in 1:n) {
  pr.Allknn = R2(knn.list.pr[[i]], cl_maxmindf.v3.v2.test$RR)*100
  knn.list.r2[[i]]<-pr.Allknn
  rm(pr.Allknn)}
knn.list.rmse <- list()
for (i in 1:n) {
  pr.Allknn = RMSE(knn.list.pr[[i]], cl_maxmindf.v3.v2.test$RR)
  knn.list.rmse[[i]]<-pr.Allknn
  rm(pr.Allknn)}
knn.list.mae <- list()
for (i in 1:n) {
  pr.Allknn = MAE(knn.list.pr[[i]], cl_maxmindf.v3.v2.test$RR)
  knn.list.mae[[i]]<-pr.Allknn
  rm(pr.Allknn)}

RMSE(knn.list.pr[[35]], cl_maxmindf.v3.v2.test$RR)
MAE(knn.list.pr[[35]], cl_maxmindf.v3.v2.test$RR)
R2(knn.list.pr[[35]], cl_maxmindf.v3.v2.test$RR)*100

# Neural networks
list.nn<-list()
k<-10
size.seq <- seq(1,k,1)
decay.seq <-as.list(seq(0.001,0.01,0.001))
for (i in 1:k) {
nn.All <- nnet(RR ~ ., data=cl_maxmindf.v3.v2, size=10, decay=decay.seq[[i]])
list.nn[[i]]<-nn.All
  rm(nn.All)}
```

```
list.nn.pr<-list()
size.seq <- seq(1,k,1)
#decay.seq <-as.list(seq(0.1,)
for (i in 1:k) {
  nn.All <- predict(list.nn[[i]], cl_maxmindf.v3.v2.test)
  list.nn.pr[[i]]<-nn.All
  rm(nn.All)}
knn.list.ann.r2 <- list()
for (i in 1:n) {
  pr.Allnn = R2(list.nn.pr[[i]], cl_maxmindf.v3.v2.test$RR)*100
  knn.list.ann.r2[[i]]<-pr.Allnn
  rm(pr.Allnn)}
knn.list.ann.rmse <- list()
for (i in 1:n) {
  pr.Allnn = RMSE(list.nn.pr[[i]], cl_maxmindf.v3.v2.test$RR)
  knn.list.ann.rmse[[i]]<-pr.Allnn
  rm(pr.Allnn)}
knn.list.ann.mae <- list()
for (i in 1:n) {
  pr.Allnn = MAE(list.nn.pr[[i]], cl_maxmindf.v3.v2.test$RR)
  knn.list.ann.mae[[i]]<-pr.Allnn
  rm(pr.Allnn)}

# Beta zero and one inflated regression
m.gam.All <- gamlss(RR~pl12m+iavg+ colli+dt1+dt2+ls1+ls2+al12m+dbal+
                age+RR.l12m+LGDpre+ctot+cavg+nl12m,
                sigma.formula = ~pl12m+iavg+ colli+dt1+dt2+ls1+ls2+al12m+dbal+
                  age+RR.l12m+LGDpre+ctot+cavg+nl12m,
                nu.formula = ~pl12m+iavg+ colli+dt1+dt2+ls1+ls2+al12m+dbal+
                  age+RR.l12m+LGDpre+ctot+cavg+nl12m,
```

```
                tau.formula = ~pl12m+iavg+ colli+dt1+dt2+ls1+ls2+al12m+dbal+
                  age+RR.l12m+LGDpre+ctot+cavg+nl12m,
                data = cl_maxmindf.v3.v2, family=BEINF)

m.gam.All <- gamlss(RR~1,data = cl_maxmindf.v3.v2, family=BEINF)
# variable selection
step_analysis <- stepGAICAll.A(m.gam.All,scope=list(lower=~1,upper=~pl12m+iavg+
colli+dt1+dt2+ls1+ls2+al12m+dbal+age+RR.l12m+LGDpre+ctot+cavg+nl12m))
# model with important variables
m.gam<-gamlss(RR~pl12m+iavg+ colli+dt1+dt2+ls1+ls2+al12m+dbal+
age+RR.l12m+LGDpre+ctot+cavg+nl12m,sigma.formula = ~
RR.l12m+pl12m+ls1+LGDpre+ls2+nl12m, nu.formula =
~pl12m+age+ls2+pl12m+dt1+RR.l12m+LGDpre+ls1+iavg+dbal,tau.formula = ~
nl12m+ls1+dbal+dt1+LGDpre+pl12m+RR.l12m , mu.formula =
~ls1+LGDpre+age+pl12m+dt1+ctot+ls2,data = cl_maxmindf.v3.v2, family=BEINF)
pr.All.a <- predictAll(m.gam, newdata =  cl_maxmindf.v3.v2.test,
type="response")%>%as.data.frame()
mean.pr.1 <- pr.All.a%>%mutate(mean.all = (tau+mu)/(1+nu+tau))
prbetaall<-mean.pr.1$mean.all%>%as.data.frame()%>%rename(RR=".")%>%
mutate(Type=paste("Predicted"))%>%
RMSE(mean.pr.1$mean.all, cl_maxmindf.v3.v2.test$RR)
MAE(mean.pr.1$mean.all, cl_maxmindf.v3.v2.test$RR)
R2(mean.pr.1$mean.all, cl_maxmindf.v3.v2.test$RR)*100

# XGboost

cl_maxmindf_0_1 <-cl_maxmindf.v3.v2
cl_maxmindf_test_0_1 <-cl_maxmindf.v3.v2.test

X_train = xgb.DMatrix(as.matrix(cl_maxmindf.v3.v2 %>% select(-RR)))
```

```
y_train = cl_maxmindf.v3.v2$RR
X_test = xgb.DMatrix(as.matrix(cl_maxmindf.v3.v2.test %>% select(-RR)))
y_test = cl_maxmindf.v3.v2.test$RR
xgb_trcontrol = trainControl(method = "cv",number = 5, allowParallel = TRUE,
  verboseIter = TRUE, returnData = FALSE
)
xgbGrid <- expand.grid(nrounds = c(50,100,200),max_depth = c(3,4,10, 15, 20,
25),colsample_bytree = seq(0.5, 0.9, length.out = 5),
eta = c(0.1, 0.2, 0.3, 0.4, 0.5),gamma=0, min_child_weight = 1,
                        subsample = c(0.1,0.3,0.5,0.6))
xgb_model = train(X_train, y_train,trControl = xgb_trcontrol,
  tuneGrid = xgbGrid,method = "xgbTree")
xgb <- xgboost(data = data.matrix(cl_maxmindf.v3.v2[,-1]), label =
cl_maxmindf.v3.v2$RR, eta = 0.3, max_depth = 4,  nround=25, subsample = 0.5,
colsample_bytree = 0.5, eval_metric = "rmse")
pr.xg <-predict(xgb,X_test)
RMSE(pr.xg, cl_maxmindf.v3.v2.test$RR)
MAE(pr.xg, cl_maxmindf.v3.v2.test$RR)
R2(pr.xg, cl_maxmindf.v3.v2.test$RR)*100
names <- dimnames(data.matrix(cl_maxmindf.v3.v2[,-1]))[[2]]
importance_matrix <- xgb.importance(names, model = xgb)

# two-stage

cl_maxmindf_0_1 <-cl_maxmindf.v3.v2%>%mutate(RR=ifelse(RR==0,0,1))
cl_maxmindf_test_0_1 <-cl_maxmindf.v3.v2.test%>%mutate(RR=ifelse(RR==0,0,1))
log_0_1 <- glm(RR ~ .,data = cl_maxmindf_0_1, family = binomial(link = "logit"))
summary(log_0_1)
# dropping variables
cl_maxmindf_0_1 <-cl_maxmindf.v3.v2%>%mutate(RR=ifelse(RR==0,0,1))%>%
```

```
  select(RR )#select(-Debt.Age, -Debtor.Type)%>%
log_0_1 <- glm(RR ~ .,data = cl_maxmindf_0_1%>%select(-al12m,-ctot,
-cavg, -dt1,-dt2,-colli), family = binomial(link = "logit"))
summary(log_0_1)
# Testing on test data
# Confusion table :
RR <- data.table(RR_0_1 = predict(log_0_1, cl_maxmindf_test_0_1,
type='response')) %>%
  .[, RR := as.factor(ifelse(RR_0_1<0.15, '0', '1'))]
cl_maxmindf_test_char<-cl_maxmindf_test_0_1%>%mutate(RR_0_1 =
as.factor(ifelse(RR>0, "1", "0")))
confusion <- confusionMatrix(table(Actual = cl_maxmindf_test_char$RR_0_1,
Prediction = RR$RR ))
confusion$table
prop.table(confusion$table, margin = 1)*100
confusion$overall['Accuracy']
model_null <- glm(RR ~ 1, data = cl_maxmindf_0_1, family = 'binomial')
anova(log_0_1, model_null, test = "Chisq")
# Negelkerke's R squared
PseudoR2(log_0_1)
cook_points <- cooks.distance(log_0_1)
influantial_points <- (cook_points>stat) %>% which()
influantial_points %>% length()
Anova(log_0_1, type="II", test="Wald")
coefs <- coef(log_0_1)
exp(coefs)

# random forests
cl_maxmindf_0_1 <-cl_maxmindf.v3.v2%>%mutate(RR=as.factor(ifelse(RR==0,"0","1"))
cl_maxmindf_test_0_1 <-cl_maxmindf.v3.v2.test%>%mutate(RR=as.factor
```

```
(ifelse(RR==0,"0","1")))
x=cl_maxmindf_0_1[,-1]
y=cl_maxmindf_0_1[,1]
tune.RF.All2 <- tuneRF(x=x, y=y, mtryStart = 2, ntree = 1000, stepFactor =
1.5,improve=1e-5,plot = TRUE,  trace = TRUE)
rfAlltune.2 <- randomForest(x=x, y=y, mtry=2, ntree = 1000, importance=TRUE)
varimp1<-varImp(rfAlltune.2)%>%as.data.frame()
varimp1$varnames <- rownames(varimp1)
varimp1<-varimp1%>%rename(Overall=‘0‘)
pr.AllRF2 <- predict(rfAlltune,newdata = pre.test)
rf.cl <- randomForest(RR~. , data = cl_maxmindf_0_1, mtry=2,
                      importance=TRUE, ntree=1000,
                      #cutoff=c(1/6,1/4),
                      classwt=c("1"=0.36,"0"=0.64))
redict.rf <- predict(rf.cl, cl_maxmindf_test_0_1)%>%as.data.frame()
predict.krf<-cbind(redict.rf, cl_maxmindf_test_0_1)%>%rename(pr.0 = ’.’)
tbl2 <- prop.table(table(predict.krf$RR,
                         predict.krf$pr.0), margin=1)*100
confusion2 <- confusionMatrix(table(predict.krf$RR,
                                    predict.krf$pr.0) )
confusion2$table
confusion2$overall[’Accuracy’]
cl_maxmindf_test_pred_1<- predict.krf%>%filter(pr.0 ==1)

# second stage. regression.
cl_maxmindf.v2_1 <-cl_maxmindf.v3.v2%>%filter(RR>0)
loan.ids <- t
cl_maxmindf_test_pred_0 <-
cl_maxmindf.v3.v2.test%>%cbind(predict.krf%>%select(pr.0))%>%
  bind_cols(loan.ids)%>%
```

```
  filter(pr.0=='0')%>%select(-pr.0)
cl_maxmindf_test_pred_1 <-
cl_maxmindf.v3.v2.test%>%cbind(predict.krf%>%select(pr.0))%>%
  bind_cols(loan.ids)%>%
  filter(pr.0=='1')%>%select(-pr.0)
vv<-cl_maxmindf_test_pred_1%>%mutate(cfs=RR*Def.Balance)
vv2<-cl_maxmindf_test_pred_0%>%mutate(cfs=RR*Def.Balance)
# prediction for non0s
pre.RF_0_1 <- cl_maxmindf.v2_1[,-1]
res.RFAll_0_1 <- cl_maxmindf.v2_1[,1]
tune.RF_1 <- tuneRF(x=pre.RF_0_1, y=res.RFAll_0_1,
mtryStart = 2, ntreeTry =1000, stepFactor = 1.5,improve=1e-5,
plot = TRUE, trace = TRUE)
rfAlltune_1 <- randomForest(x=pre.RF_0_1, y=res.RFAll_0_1,
mtry=6, ntree = 1000,importance=TRUE)
varimp1<-varImp(rfAlltune_1)%>%as.data.frame()
varimp1$varnames <- rownames(varimp1)
pre.test <- cl_maxmindf_test_pred_1[,c(-1,-17)]
pr.AllRF <- predict(rfAlltune_1,newdata = pre.test)
RMSE(pr.AllRF, cl_maxmindf_test_pred_1$RR)
MAE(pr.AllRF, cl_maxmindf_test_pred_1$RR)
R2(pr.AllRF, cl_maxmindf_test_pred_1$RR)*100
error.check <- plot.data%>%filter(Type=="Predicted")%>%select(RR)
error.check2 <- plot.data%>%filter(Type=="Actual")%>%select(RR)
RMSE(error.check$RR, error.check2$RR)
MAE(error.check$RR, error.check2$RR)
R2(error.check$RR, error.check2$RR)*100
# KNN
pre.RF_0_1 <- cl_maxmindf.v2_1[,-1]
res.RFAll_0_1 <- cl_maxmindf.v2_1[,1]
```

```
pre.test <- cl_maxmindf_test_pred_1[,-1]
knn.list <- list()
n<-40
for (i in 1:n) {
  fit_knn_All = knnreg(x = pre.RF_0_1, y = res.RFAll_0_1, k=i )
  knn.list[[i]]<-fit_knn_All
  rm(fit_knn_All)}
knn.list.pr <- list()
for (i in 1:n) {
  pr.Allknn = predict(knn.list[[i]],newdata = pre.test[,-16])
  knn.list.pr[[i]]<-pr.Allknn
  rm(pr.Allknn)}
knn.list.r2 <- list()
for (i in 1:n) {
  pr.Allknn = R2(knn.list.pr[[i]], pre.test$RR)*100
  knn.list.r2[[i]]<-pr.Allknn
  rm(pr.Allknn)}

knn.list.rmse <- list()
for (i in 1:n) {
  pr.Allknn = RMSE(knn.list.pr[[i]], pre.test$RR)
  knn.list.rmse[[i]]<-pr.Allknn
  rm(pr.Allknn)}
knn.list.mae <- list()
for (i in 1:n) {
  pr.Allknn = MAE(knn.list.pr[[i]], pre.test$RR)
  knn.list.mae[[i]]<-pr.Allknn
  rm(pr.Allknn)}
knn.list.PT <- list()
for (i in 1:n) {
```

```
  prknnall<-knn.list.pr[[i]]%>%as.data.frame()
  plot.data2<-cl_maxmindf_test_pred_1%>%select(RR,
  Acq.Balance)%>%bind_cols(prknnall)%>%rename(Pr=".")%>%
    bind_rows(cl_maxmindf_test_pred_0%>%select(RR, Acq.Balance)%>%mutate(Pr=0))
pr.Allknn = predict(knn.list[[30]],newdata = pre.test[,-16])
pr.Allknn<-pr.Allknn%>%as.data.frame()%>%rename(Pr=".")%>%
mutate(Type=paste("Predicted"))%>%
  bind_cols(cl_maxmindf_test_pred_1%>%select(Acq.Balance,RR))%>%
  bind_rows(cl_maxmindf_test_pred_0%>%mutate(Type=paste("Predicted"),
  Pr=0)%>%select(RR,Pr,Type,Acq.Balance))
true.1 <-pr.Allknn%>%select(RR)%>%mutate(Type="Actual")
pred.1 <-pr.Allknn%>%select(Pr)%>%mutate(Type="Predicted",RR=Pr)
error.check <- plot.data%>%filter(Type=="Predicted")%>%select(RR)
error.check2 <- plot.data%>%filter(Type=="Actual")%>%select(RR)
RMSE(error.check$RR, error.check2$RR)
MAE(error.check$RR, error.check2$RR)
R2(error.check$RR, error.check2$RR)*100

# neural networks

list.nn<-list()
k<-10
size.seq <- seq(1,k,1)
decay.seq <-as.list(seq(0.001,0.01,0.001))
for (i in 1:k) {
  nn.All <- nnet(RR ~ ., data=cl_maxmindf.v2_1, size=10, decay=decay.seq[[i]])
  list.nn[[i]]<-nn.All
  rm(nn.All)}
list.nn.pr<-list()
size.seq <- seq(1,k,1)
```

```
#decay.seq <-as.list(seq(0.1,)
for (i in 1:k) {
  nn.All <- predict(list.nn[[i]], pre.test)
  list.nn.pr[[i]]<-nn.All
  rm(nn.All)}
knn.list.ann.r2 <- list()
n<-k
for (i in 1:n) {
  pr.Allnn = R2(list.nn.pr[[i]], pre.test$RR)*100
  knn.list.ann.r2[[i]]<-pr.Allnn
  rm(pr.Allnn)}
knn.list.ann.rmse <- list()
for (i in 1:n) {
  pr.Allnn = RMSE(list.nn.pr[[i]], cl_maxmindf_test_pred_1$RR)
  knn.list.ann.rmse[[i]]<-pr.Allnn
  rm(pr.Allnn)}
knn.list.ann.mae <- list()
for (i in 1:n) {
  pr.Allnn = MAE(list.nn.pr[[i]], cl_maxmindf_test_pred_1$RR)
  knn.list.ann.mae[[i]]<-pr.Allnn
  rm(pr.Allnn)}
pr.Allnn = predict(list.nn[[1]],newdata = pre.test[,-16])
error.check <- plot.data%>%filter(Type=="Predicted")%>%select(RR)
error.check2 <- plot.data%>%filter(Type=="Actual")%>%select(RR)
RMSE(error.check$RR, error.check2$RR)
MAE(error.check$RR, error.check2$RR)
R2(error.check$RR, error.check2$RR)*100
plotnet(list.nn[[3]])
variable importance plot
g2<-garson(list.nn[[1]])
```

```
#SVMs
SVM.2 <- svm(RR ~ ., data = cl_maxmindf.v2_1, kernel = "radial",
             cost=0.2, gamma=0.1)
SVM.22 <- svm(RR ~ ., data = cl_maxmindf.v2_1, kernel = "sigmoid",
             cost=0.002, nu=0.01)
pre_SVM.test <- predict(SVM.2,
cl_maxmindf_test_pred_1[,2:16])%>%as.data.frame()%>%rename(Pr=".")
pre_SVM.test2 <- predict(SVM.22,
cl_maxmindf_test_pred_1[,2:16])%>%as.data.frame()%>%rename(Pr=".")
prnsvm<-pre_SVM.test%>%mutate(Type=paste("Predicted"))%>%
  bind_cols(cl_maxmindf_test_pred_1%>%select(Acq.Balance,RR))%>%
  bind_rows(cl_maxmindf_test_pred_0%>%mutate(Type=paste("Predicted"),
  Pr=0)%>%select(RR,Pr,Type,Acq.Balance))
true.1 <-prnsvm%>%select(RR)%>%mutate(Type="Actual")
pred.1 <-prnsvm%>%select(Pr)%>%mutate(Type="Predicted",RR=Pr)
error.check <- plot.data%>%filter(Type=="Predicted")%>%select(RR)
error.check2 <- plot.data%>%filter(Type=="Actual")%>%select(RR)
RMSE(error.check$RR, error.check2$RR)
MAE(error.check$RR, error.check2$RR)
R2(error.check$RR, error.check2$RR)*100
# xgboost

xgb.2 <- xgboost(data = data.matrix(cl_maxmindf.v2_1[,-1]),
label = cl_maxmindf.v2_1$RR, eta = 0.3, max_depth = 5,  nround=25,
subsample = 0.5,colsample_bytree = 0.5,val_metric = "rmse",
objective = "reg:linear")
pre_xgb.test <- predict(xgb.2,xgb.DMatrix(as.matrix(
cl_maxmindf_test_pred_1[,2:16])))%>%as.data.frame()%>%rename(Pr=".")
prnxgb<-pre_xgb.test%>%mutate(Type=paste("Predicted"))%>%
    bind_cols(cl_maxmindf_test_pred_1%>%select(Acq.Balance,RR))%>%
```

```
    bind_rows(cl_maxmindf_test_pred_0%>%mutate(Type=paste("Predicted"),
 Pr=0)%>%select(RR,Pr,Type,Acq.Balance))
  error.check <- plot.data%>%filter(Type=="Predicted")%>%select(RR)
  error.check2 <- plot.data%>%filter(Type=="Actual")%>%select(RR)
  RMSE(error.check$RR, error.check2$RR)
  MAE(error.check$RR, error.check2$RR)
  R2(error.check$RR, error.check2$RR)*100
x2<-xgb.importance(names, model = xgb.2)
x2<-xgb.plot.importance(importance_matrix[1:10,])

# ensemble
s_size2 <- floor(0.5 * nrow(cl_maxmindf.v3.v2))
t_ind2 <- sample(seq_len(nrow(cl_maxmindf.v3.v2)), size = s_size2)
train.ensemble<-cl_maxmindf.v3.v2[t_ind2,]
test.ensemble <- cl_maxmindf.v3.v2[-t_ind2,]
x.test <- test.ensemble[,-1]
x2<-train.ensemble[,-1]
y2<-train.ensemble[,1]
rf.ensemble <- randomForest(x=x2, y=y2, mtry=3, ntree = 800, importance=TRUE)
nnet.ensemble <- nnet(RR~., train.ensemble, size=10, decay=0.003)
knn.ensemble <- knnreg(x = x2, y = y2, k=35 )
xgb_trcontrol = trainControl(method = "cv",number = 5,allowParallel = TRUE,
  verboseIter = TRUE, returnData = FALSE)
xgbGrid <- expand.grid(nrounds = c(50,25),max_depth = c(3,4,10),
colsample_bytree = seq(0.5, 0.9, length.out = 5),eta = c(0.01, 0.1, 0.3),
gamma=0,min_child_weight = 1,subsample = c(0.1,0.2,0.5))
xgb_model = train(
  data.matrix(train.ensemble[,-1]), train.ensemble$RR,
  trControl = xgb_trcontrol, tuneGrid = xgbGrid,method = "xgbTree")
knn.ensemble <- xgboost(data = data.matrix(train.ensemble[,-1]),
```

```
label = train.ensemble$RR, eta = 0.1,max_depth = 3,  nround=50,
subsample = 0.5,colsample_bytree = 0.8,
eval_metric = "rmse", objective = "reg:linear")

# predictions
rf.ensemble.pred <- predict(rf.ensemble, newdata = x.test)
nnet.ensemble.pred <- predict(nnet.ensemble, newdata = x.test)
knn.ensemble.pred <- predict(knn.ensemble, newdata = as.matrix(x.test))
predDF.ensemb <- data.frame(rf.ensemble.pred,
                            nnet.ensemble.pred,
                            knn.ensemble.pred,
                            RR = test.ensemble$RR)
x3<-predDF.ensemb[,-4]
y3<-predDF.ensemb[,4]
modelStack.ensemble<-randomForest(x=x3, y=y3, mtry=2, ntree = 800,
importance=TRUE)
as.matrix(cl_maxmindf.v3.v2.test[,-1]))
predDF.ensemb.test <- data.frame(rf.ensemble.test,
                                 nnet.ensemble.test,
                                 knn.ensemble.test,
                            RR = cl_maxmindf.v3.v2.test$RR)%>%
  rename(rf.ensemble.pred=rf.ensemble.test,
         nnet.ensemble.pred=nnet.ensemble.test,
         knn.ensemble.pred=knn.ensemble.test)
combPred.ens <- predict(modelStack.ensemble, as.matrix(predDF.ensemb.test[,-4]))
R2(combPred.ens,cl_maxmindf.v3.v2.test$RR)*100
RMSE(combPred.ens,cl_maxmindf.v3.v2.test$RR)
MAE(combPred.ens,cl_maxmindf.v3.v2.test$RR)

#monthly forecast
```

```
A<-Accounts[train_ind,]
B<-Accounts[-train_ind,]
loop.train<-cl_maxmindf.v3.v2%>%select(-RR)%>%bind_cols(A[,37:64]%>%
mutate_if(is.numeric, ~round(., 2)))
loop.test<-cl_maxmindf.v3.v2.test%>%select(-RR)%>%bind_cols(B[,37:64]%>%
mutate_if(is.numeric, ~round(., 2)))
# predictors for tests
loop.test.predictors <- loop.test[,1:15]
list_predict <- list()
tune.out <- list()
for (i in 1:28) {
  list_model <- list()
  y = loop.train[,15+i]#%>%as.data.frame()%>%rename(M=".")
  x = loop.train[,c(1:15)]
  list_model[[i]] <- randomForest(x=x, y=y, mtry=3, ntree=500)
  list_predict[[i]] <- predict(list_model[[i]], loop.test.predictors)
  rm(list_model)}
predict_total <- as.data.frame(bind_cols(list_predict))%>%
bind_cols(as.data.frame(loop.test[,c(16:43)]))%>%
  mutate_if(is.numeric, ~round(., 2))%>%
  bind_cols(t%>%select(Acq.Balance))%>%rename(Def.Balance=Acq.Balance)%>%
  summarise(
    FC_1 = sum(V1*Def.Balance)/sum(t$Acq.Balance),
    ...
    FC_28 = sum(V28*Def.Balance)/sum(t$Acq.Balance),
    Act_1 = sum(M1*Def.Balance)/sum(t$Acq.Balance),
    ...
    Act_28 = sum(M28*Def.Balance)/sum(t$Acq.Balance))%>%as.data.frame()
```