Thomas Benjamin Frogner

Learning from Imbalanced Data, with a Case Study in Finance

Master's thesis in Applied Physics and Mathematics
Supervisor: John Sølve Tyssedal
August 2019

Thomas Benjamin Frogner

Learning from Imbalanced Data, with a Case Study in Finance

Master’s thesis in Applied Physics and Mathematics
Supervisor: John Sølve Tyssedal
August 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Abstract

This thesis investigates some of the challenges with classifying imbalanced data sets, and logistic regression and random forests are the chosen algorithms. The main issue to deal with when classifying imbalanced data is the fact that most algorithms tend to maximize accuracy, which means that few of the minority class observations are correctly classified. Resampling techniques like undersampling, oversampling and SMOTE were used to balance the data set in order to improve the performance of the classifiers. A simplified version of SMOTE was implemented and used here. The performance was measured by balanced accuracy, and all three resampling methods seemed to yield quite similar results in many cases. For random forests, undersampling seemed to behave differently from oversampling and SMOTE. When applied to logistic regression, the resampling techniques all performed quite evenly.

A data set was provided by Sparebank 1 Kredittkort AS, with the aim of identifying which customers might seek refinancing of their credit card debt from competing banks. Undersampling increased the balanced accuracy of random forests on this data set from 0.77 to 0.82. Logistic regression achieved a balanced accuracy of roughly 0.79 with no resampling, and 0.81 with resampling. Oversampling and SMOTE appeared to be slightly more effective on logistic regression than undersampling.

Sammendrag

Denne avhandlingen undersøker noen av utfordringene ved å klassifisere et ubalansert datasett, og de valgte algoritmene er logistisk regresjon og random forests (en metode basert på valgtrær). Hovedproblemet å håndtere når man skal klassifisere ubalanserte datasett er det faktum at de fleste algortimer har som tendens å maksimere nøyaktigheten, som betyr at få av observasjonene i minoritetsklassen blir klassifisert på riktig vis. Resamplingsteknikker som undersampling, oversampling og SMOTE - syntetisk minoritets oversamplings teknikk - ble brukt for å balansere datasettet for å forbedre ytelsen til klassifikasjonsalgoritmene. En forenklet versjon av SMOTE ble implementert og brukt her. Ytelsen ble målt ved balansert nøyaktighet, og alle tre resamplingsmetoder så ut til å gi veldig like resultater i mange tilfeller. For random forests så undersampling ut til å oppføre seg ganske annerledes enn oversampling og SMOTE. Resamplingsteknikkene så ut til å yte svært likt da de ble brukt på logistisk regresjon.

Et datasett ble gitt av Sparebank 1 Kredittkort AS, med mål om å identifisere hvilke kunder som kunne komme til å søke refinansiering av kredittkortgjelden sin hos konkurrerende banker. Undersampling økte den balanserte nøyaktigheten til random forests fra 0.77 til 0.82. Logistisk regresjon oppnådde en balansert nøyaktighet på omtrent 0.79 uten resampling, og 0.81 med resampling. Oversampling og SMOTE så ut til å være noe mer effektive på logistisk regresjon enn undersampling.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) in the field of statistical learning. The thesis was written at the Department of Mathematical Sciences during the spring and summer of 2019, in cooperation with Sparebank 1 Kredittkort AS. It is assumed that the reader has some knowledge of statistics, particularly logistic regression and classificaton.

I would like to extend my thanks to Christian Meland and Jens Morten Nilsen at Sparebank 1 Kredittkort AS for the opportunity in writing this thesis. My supervisor John Sølve Tyssedal has been supportive and patient with me throughout the past year, and I thank him for his detailed and swift feedback to any questions I have had.

Furthermore, I would like to thank my parents for their never-ending support, and for my classmates who have made these past five years an amazing time in my life.

Table of Contents

Chapter 1

Introduction

Today, banks have several major sources of income, credit cards being one of them. In the US, average annual percentage rates are at one of their highest values ever of 17.71%, as of August 22, 2019 (Dilworth, 2019). In addition to interest on unpaid balance, some companies charge fees on the credit cards that can accumulate to more than 100 USD annually. As of February 2019, the Norwegian government has imposed more strict regulation regarding consumer loans and credit cards, in order to reduce the amount of unsecured debt in society (Finansdepartementet, 2019).

It sometimes happens that a customer is not able to pay off their unpaid balance. Refinancing of this debt can be sought at a competing bank, making the original bank lose out on the profits from this customer. If we are able to determine who will seek refinancing from competitors, then the original bank has the opportunity of offering refinancing of the debt before the customer is lost to a competitor, thereby retaining some of the interest on the debt that is owed. This problem is one we will attempt to solve on behalf of Sparebank 1 Kredittkort AS, who have graciously provided data on more than 600 000 customers, and this was the main topic for the project thesis that was finished in March of this year (Frogner, 2019). Here, we will take a slightly more general approach and study imbalanced data in more detail as the data set provided is highly imbalanced, with less than 0.5% of the customers seeking refinancing. We will attempt to sample data from known distributions, and see if the findings on these data correspond to the findings on the data set from Sparebank 1.

When attempting to classify the data set provided, we will be using logistic regression and random forests. Logistic regression is a well known approach within classification problems with binary outcomes, while random forests are based off of decision trees and have not been used by statisticians as much. Logistic regression is considered to be a more standard approach from a statisticians point of view. Decision trees are thought to mimic human behavior, and for this reason they are considered to be a good alternative when trying to predict what a customer might do. Decision trees also have a less rigid mathematical structure than logistic regression. Therefore, it will be interesting to see which has the better predictive performance of random forests and logistic regression.

Chapter 2

Theory

This section will go through theory that is relevant to classification on imbalanced data sets, and on classification in general. The methods that will be covered are logistic regression and random forests, the latter being a method based on decision trees. Logistic regression will be the primary method, as this is a method that is well understood by statisticians.

Throughout this chapter we will refer to the target variable Y as the response, which can take on the values 0 or 1. An observation where $Y = 0$ is referred to as a negative observation, and when $Y = 1$ we refer to it as a positive observation. Index i refers to observation number i, and the covariates of Y_i are found in the k-dimensional vector $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ik})^T$. Sometimes a "1" will be added to this vector, when necessary.

As this thesis was inspired by my project thesis (Frogner, 2019) a lot of the theory presented here is similar to that of the project. Some of the figures that are used to illustrate the theory are taken directly from the project thesis.

2.1 Classification

Given an instance of a binary variable we have four possible outcomes: we can correctly classify a negative response (true negative), we can incorrectly classify a negative response as positive (false positive), we can incorrectly classify a positive response as negative (false negative) or we can correctly classify a positive response (true positive). For a given data set we can summarize the number of each of these in a confusion matrix, as per table 2.1. Obviously, we wish to maximize the true negative rate (TNR) and true positive rate (TPR)

	Prediction: 0	Prediction: 1
Reference: 0	True Negative (TN)	False Positive (FP)
Reference: 1	False Negative (FN)	True Positive (TP)

Table 2.1: A confusion matrix for visualizing the performance of a classification method.

simultaneously, which are defined by

$$\text{TNR} = \frac{\text{TN}}{\text{TN + FP}}, \quad \text{and} \quad \text{TPR} = \frac{\text{TP}}{\text{TP + FN}}. \tag{2.1}$$

For a model that attempts to classify an imbalanced data set we expect the true negative rate to be high and the true positive rate to be low, as most models will classify almost all the observations to the minority class, including many of the positive observations. This is due to many algorithms attempting to optimize accuracy (Rahman and Davis, 2013), which is easily done by classifying to the majority class.

2.1.1 The Performance Measure

In order to determine what is a good classifier we must specify a suitable performance measure. We start with one of the most intuitive loss functions, namely 0/1 Loss.

0/1 Loss

Given an observation Y_i, we define the 0/1 loss by

$$\text{Loss}_i := \begin{cases} 0 & \text{if case } i \text{ is correctly classified,} \\ 1 & \text{otherwise.} \end{cases} \tag{2.2}$$

This should be averaged over all n observations, yielding the error rate:

$$\text{Error Rate} = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}_i, \tag{2.3}$$

which should be well known even to a layman. The accuracy of a model is 1 minus this error rate. Intuitively one might choose the accuracy of a classifier as the performance measure, but in the case of classifying imbalanced data this would be a mistake. Consider a data set with 99 % of the cases being negative observations. If we were to classify every single observation as negative we would achieve an accuracy of 99 %. An accuracy this high would often be considered to be quite good, while in this case it is entirely useless. For imbalanced data sets we must therefore use some other performance measure.

Balanced Accuracy

The balanced accuracy (BACC) of a classifier is defined by

$$\text{BACC} = \frac{1}{2}(\text{True Negative Rate + True Positive Rate}) \tag{2.4}$$

and is considered to be an excellent performance measure when working with imbalanced data sets (Lujan-Moreno et al., 2018). The balanced accuracy has the advantage of rewarding methods that are able to correctly classify some of the observations from the minority class, while still keeping the accuracy up. We can easily generalize the BACC to place uneven weight on the true negative and true positive rates:

$$\text{BACC}_\lambda := (1-\lambda)\cdot\text{TNR} + \lambda\cdot\text{TPR}, \quad \lambda \in (0,1). \tag{2.5}$$

This generalization allows for more flexibility than the first definition and the definitions are equal when $\lambda = 1/2$. This performance measure is especially good when we have a classification problem where a false negative and a false positive are of uneven concern, i.e a false negative might be undesirable in a medical setting, as we then miss out on a sick patient who goes un-diagnosed.

Brier Score

The Brier Score (BS) (Brier, 1950) is defined by

$$\text{BS} = \frac{1}{n}\sum_{i=1}^{n}(p_i - Y_i)^2, \tag{2.6}$$

where p_i is the predicted probability that $Y_i = 1$. We can see that this is the mean squared error of the predictor. This loss function was first formulated by Glenn W. Brier and was used to measure the precision of weather reports. The rationale was that uncertain predictions should be penalized. If rain is observed on a particular day in a particular location then it would have been better if a 90 % chance of rain had been predicted, as opposed to a 70 % chance of rain.

While it is clear that this loss function has its use, we will not be utilizing it here, as the balanced accuracy should be a sufficient measure of which model is better. Which loss function is appropriate to use is a somewhat subjective problem to determine and there are probably many other loss functions that could be discussed here. The generalized version of the balanced accuracy allows for a lot of flexibility, and the mixing parameter λ can be chosen to accommodate any wishes one might have for the classifier. In some cases it could be very important to correctly classify almost all the positive observations, and we might therefore pick $\lambda > 1/2$. An example for this could be some screening for a rare disease. The false positive rate will increase, but this can be dealt with through further testing, and the goal of correctly determining who has the disease is maintained.

2.1.2 Training, Validating and Testing

When fitting a model to a data set, one generally needs to first split the data set into a training set, a validation set and a test set. In order not to overfit the data we fit the candidate models on the training set and compare their performance on the validation set. The model that appears to be best according to the desired criterion is selected based on this performance, and then this model is applied to the test set in order to report an unbiased estimate of the performance of the model.

In this thesis we will be comparing methods to improve classification and the goal is to determine if one method is better than another. We will mainly be using the balanced accuracy of a classifier to evaluate what constitutes a good learning algorithm, and we care only about the relative performances of two separate methods, not their absolute performance. For this reason we can skip the step of using a validation set and ignore the fact that the performance measure might be slightly biased, as we are picking the model that we know has the best performance measure and using this on the test set. This performance measure will then probably a bit better than an unbiased estimate, which is unproblematic as we are simply comparing models.

When it comes to evaluating the performance of random forests, validation sets are never needed, due to the nature of how these are created. For details, see section 2.3.2.

2.1.3 Undersampling

We now move on to a common method in dealing with imbalanced data sets, namely undersampling. Undersampling works by intentionally omitting a large portion of observations from the majority class in the training set, thereby increasing the proportion of minority cases in the training set. The idea is that an algorithm might be able to perform better when it is trained on a less imbalanced data set.

Given a data set of size n with a fraction p of the observations being positive we can decide to train our algorithm on a training set with an equal amount of negative and positive observations. After omitting sufficiently many negative observations this training set will consist of a total of $2np$ observations in order to become perfectly balanced. There are (at least) three concerns regarding this:

1. As $p \ll 1$, this will be a dramatic reduction in sample size. If data is plentiful this might not be an issue, but it is rarely desirable to reduce the number of available observations. It is entirely possible that a classifier trained on such a training set will miss out on valuable information regarding what classifies a negative observation.

2. If some of the covariates in the data set are categorical then they may have levels that don't show up in the training set. Methods like logistic regression are not meant to deal with this issue. One solution could be to use different random seeds when choosing the training set to combat this issue, but this is an inelegant solution that might not always work. This issue was encountered in my project thesis (Frogner, 2019), and the chosen solution was to limit the degree of undersampling so that the training set consisted of up to no more than 20 % positive observations, thereby limiting the reduction of the sample size.

3. The third concern is that the chosen algorithm might become too eager to classify to the majority class, as it is trained on a very biased data set. Note that "too eager" depends on the choice of loss function. Latinne et al. (2001) proposed an adjustment of the output of a classification algorithm in order to correct for this bias and this is discussed in section 2.1.5.

2.1.4 Oversampling & Synthetic Minority Oversampling Technique

In this section we will be discussing two quite similar methods for creating a more balanced data set. The first is oversampling, which is to draw observations from the minority class additional times in order to make the data set less imbalanced. We can either decide to copy each of the positive observations a certain number of times, or we can randomly draw from these observations with replacement until we are satisfied with the balance of the training set.

The second method is called Synthetic Minority Oversampling Technique (SMOTE) and works by randomly taking two observations from the minority class and creating a new, synthetic data point on the line segment joining these observations (Chawla et al., 2002). The new data point will be given the same response as the two points we sampled. For continuous variables the new sample point $\boldsymbol{x}_{\text{new}}$ is created from two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the following way:

$$\boldsymbol{x}_{\text{new}} := a\boldsymbol{x}_i + (1-a)\boldsymbol{x}_j, \quad a \in (0,1). \tag{2.7}$$

The mixing parameter a is randomly sampled from the interval (0,1). The hope is that this method can help to generalize the predictor space by introducing new points that hopefully contain information that is representative for a minority observation. Categorical variables are a slight complication to this algorithm. When two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are chosen to create the new data point, they might have categorical variables with differing levels. Chawla et al. (2002) decided that the new point would be given the factor level that was most common among the k-nearest neighbors of the new point, for some k. Beyer et al. (1999) studied the k-nearest neighbor algorithm extensively for high dimensional cases, and was able to show that in some cases the difference between the closest and farthest point approached each other as the number of dimensions grew. The Euclidian distance between two points is easy to calculate for continuous variables, but with differing factor levels one must provide some addition to the distance measure to correctly identify how far away two points lie from each other. We will not delve exactly into how this was handled by Chawla et al. (2002) as we will be taking a simpler approach; when sampling two points we will simply be giving the new observation the exact same factor levels as the first observation we sampled, $\boldsymbol{x}_i$. We do not wish to calculate the nearest neighbors, as the predictor space is sparsely inhabited when the number of predictors grows large. The data set provided by Sparebank 1 consists of a response and 77 additional variables, and we decide that using the k-nearest neighbors is not an optimal course of action. Our solution is to sample the mixing parameter a from values close to 1 and far from 0, thereby making sure that $\boldsymbol{x}_{\text{new}}$ lies closer to $\boldsymbol{x}_i$ than $\boldsymbol{x}_j$. We do not wish to have the new point far from $\boldsymbol{x}_i$, as points far away might have entirely different factor levels for their categorical variables. We then decide to sample a uniformly from the interval $[a', 1]$ for some appropriate value for a'. In this thesis we will use the interval $[0.5, 1]$ to sample a from, as this ensures that the new point lies closer to the point it inherits its factor levels from, and not to the other point that may have other factor levels.

The term "degree of over/undersampling" has been used a few times and refers to the fraction of minority observations in the training set, regardless of the fraction in the original data set. For instance, if we have an oversampled data set with 20 % of the observations belonging to the minority class, the degree of oversampling is 20 %, regardless of the imbalance in the original data set.

2.1.5 Cut-Off Probability

Assume we have a model that predicted that $Y_i = 1$ with probability p_i for observation i. We typically classify Y_i as 1 if $p_i > 1/2$, but the discriminating value need not be 1/2. Any value in the interval (0,1) will do, and we therefore adhere to the more flexible classification rule given by

$$Y_i = \begin{cases} 0 & \text{if } p_i < \alpha, \quad \alpha \in (0,1), \\ 1 & \text{otherwise.} \end{cases} \tag{2.8}$$

We refer to α as our cut-off probability, and this can be varied to increase the number of observations classified to the minority class.

Adjusting the Cut-Off Probability

When training a model on a training set that is either oversampled or undersampled, the model will become biased in favor of the minority class. Latinne et al. (2001) proposed a way to adjust the class probabilities given by a model by using Bayesian inference. We will derive the formula for this adjustment based on the derivation in Latinne et al. (2001).

We first start by specifying some notation that is needed for this derivation. We treat the general case of a C-class problem, with class labels $\omega_0, \omega_1, \ldots, \omega_{C-1}$. $p(\omega_i)$, $i = 0, 1, \ldots, C-1$ is the probability of a random observation in the original data set belonging to class ω_i, and $p_t(\omega_i)$ is the probability of a random observation in the training set belonging to class ω_i. Note that these probabilities are based only on the imbalance of their respective data sets. Subscript t is used to refer to the training set.

Bayesian inference views parameters as stochastic variables instead of fixed values. Prior distributions for the parameters are based on the observer's subjective views on what these distributions should be, and when data $\boldsymbol{x}$ is observed the distribution is updated and we end up with our posterior distribution. Bayes' theorem states that we have the following relation:

$$p(\boldsymbol{x}|\omega_i) = \frac{p(\omega_i|\boldsymbol{x})p(\boldsymbol{x})}{p(\omega_i)}. \tag{2.9}$$

Here $p(\boldsymbol{x}|\omega_i)$ is the distribution of the covariates $\boldsymbol{x}$ given that the response belongs to class ω_i, $p(\boldsymbol{x})$ is the distribution of the observation vectors for the entire population and $p(\omega_i|\boldsymbol{x})$ is what we are attempting to predict: the posterior probability that the response belongs to ω_i given data $\boldsymbol{x}$. The same relation applies to data from the training set:

$$p_t(\boldsymbol{x}|\omega_i) = \frac{p_t(\omega_i|\boldsymbol{x})p_t(\boldsymbol{x})}{p_t(\omega_i)}, \tag{2.10}$$

where $p_t(\omega_i|\boldsymbol{x})$ is the probability of Y belonging to class ω_i given by a model trained on the over/undersampled training set, given data $\boldsymbol{x}$. $p_t(\omega_i)$ simply equals the rate of over/undersampling, as stated above. $p_t(\boldsymbol{x})$ is the distribution of the observation vector evaluated at $\boldsymbol{x}$. Note that this distribution also depends upon the training set. However, the left hand side of both equations (2.9) and (2.10) does not depend upon which data set we are looking at. Given the class ω_i the distribution of the covariates $\boldsymbol{x}$ do not depend on what data set they are taken from. For this reason we can equate the right hand sides of equations (2.9) and (2.10) to solve for our target probability

$$p(\omega_i|\boldsymbol{x}) = \frac{p_t(\boldsymbol{x})}{p(\boldsymbol{x})} \frac{p(\omega_i)}{p_t(\omega_i)} p_t(\omega_i|\boldsymbol{x}). \tag{2.11}$$

Necessarily,

$$\sum_{i=0}^{C-1} p(\omega_i|\boldsymbol{x}) = 1 \tag{2.12}$$

as each observation must belong to one of the C classes. The first fraction in equation (2.11) becomes a normalizing constant to ensure that equation (2.12) holds, and we have that

$$\frac{p_t(\boldsymbol{x})}{p(\boldsymbol{x})} = \left[\sum_{i=0}^{C-1} \frac{p(\omega_i)}{p_t(\omega_i)} \cdot p_t(\omega_i|\boldsymbol{x})\right]^{-1}. \tag{2.13}$$

We now simplify to having only two classes and use that $p(\omega_0) = 1 - p(\omega_1)$, and similarly for all other probabilities. Using this and equation (2.13) and inserting into equation (2.11) we obtain our posterior, unbiased probability:

$$p(\omega_1|\boldsymbol{x}) = \frac{\frac{p(\omega_1)}{p_t(\omega_1)} \cdot p_t(\omega_1|\boldsymbol{x})}{\frac{p(\omega_1)}{p_t(\omega_1)} \cdot p_t(\omega_1|\boldsymbol{x}) + \frac{1-p(\omega_1)}{1-p_t(\omega_1)} \cdot (1 - p_t(\omega_1|\boldsymbol{x}))}. \tag{2.14}$$

If we are to use a cut-off probability of 1/2 on the unbiased probability, we can set the left hand side greater than 1/2 and solve for $p_t(\omega_1|\boldsymbol{x})$. A little bit of algebra yields

$$p_t(\omega_1|\boldsymbol{x}) > \frac{p_t(\omega_1)(1 - p(\omega_1))}{p(\omega_1)(1 - p_t(\omega_1)) + p_t(\omega_1)(1 - p(\omega_1))} \tag{2.15}$$

as our cut-off probability. We will be investigating the effect of oversampling and undersampling, and we will be using this adjustment to see if it leads to improved performance of any given model.

2.1.6 Correlation in Imbalanced Data & Mixed Distributions

Given two stochastic variables X and Y, their correlation ρ is defined by

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y}, \tag{2.16}$$

where $\mathrm{Cov}(X, Y)$ is the covariance between the variables, and σ_X and σ_Y are their respective standard deviations. We want to look at the correlation between an explanatory variable X and the response Y in an imbalanced dataset, where

$$Y \sim \mathrm{Bin}(1, p), \quad p \ll 1, \tag{2.17}$$

$$X|(Y = 0) \sim N(\mu_0, \sigma_0^2), \tag{2.18}$$

$$X|(Y = 1) \sim N(\mu_1, \sigma_1^2). \tag{2.19}$$

p is estimated by using the relative frequency of positive observations in the dataset. We have that $p \ll 1$, because the dataset is imbalanced and the majority class is set to be 0. We have also assumed that the explanatory variable X comes from a known normal distribution that depends on what the response was for the given observation. Furthermore, we can standardize a normally distributed variable to have unit variance by dividing the variable by its standard deviation, so we assume that this has been done and that $\sigma_0^2 = \sigma_1^2 = 1$.

Given n independent observational pairs $(X_i, Y_i) = (x_i, y_i), i = 1, \dots, n$, we have that the sample correlation coefficient r is given by

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y}, \tag{2.20}$$

where $\bar{x} = n^{-1}\sum_{i=1}^{n} x_i$, and similarly for $\bar{y}$. s_x^2 and s_y^2 are the unbiased sample variances. Dividing the sum into parts where $y_i = 0$ and $y_i = 1$ we can expand equation (2.20) in the following way:

$$r_{X,Y} = \frac{1}{(n-1)s_x s_y}\left[\sum_{y_i=0}(x_i - \bar{x})(0 - \bar{y}) + \sum_{y_i=1}(x_i - \bar{x})(1 - \bar{y})\right]. \tag{2.21}$$

The first sum extends over approximately $n(1-p)$ elements, and the second extends over approximately np elements. The x_i in the first sum come from a distribution with mean μ_0 and the x_i in the second sum come from a distribution with mean μ_1. We can then approximate the mean $\bar{x}$ as a weighted sum of these means:

$$\bar{x} \approx (1-p)\mu_0 + p\mu_1. \tag{2.22}$$

p is approximated to be $\bar{y}$. Inserting this into equation (2.21) we can move some terms in parenthesis outside their respective sums:

$$r_{X,Y} = \frac{1}{(n-1)s_x s_y}\left[-p\sum_{y_i=0}(x_i - \bar{x}) \ + \ (1-p)\sum_{y_i=1}(x_i - \bar{x})\right]. \tag{2.23}$$

For all i where $y_i = 0$ we know that the mean of the x_i should be approximately equal to μ_0. Subtraction by the grand mean $\bar{x}$ in each term serves only as a translation of the terms. Therefore, we can approximate

$$\begin{aligned} -p\sum_{y_i=0}(x_i - \bar{x}) &\approx -pn(1-p)(\mu_0 - (1-p)\mu_0 - p\mu_1) \\ &= np^2(1-p)(\mu_1 - \mu_0), \end{aligned} \tag{2.24}$$

where we recall that multiplication by $n(1-p)$ is due to the fact that this is the number of terms in the sum. Similarly, for the second sum in equation (2.23) we obtain

$$\begin{aligned} (1-p)\sum_{y_i=1}(x_i - \bar{x}) &\approx (1-p)np(\mu_1 - (1-p)\mu_0 - p\mu_1) \\ &= np(1-p)^2(\mu_1 - \mu_0). \end{aligned} \tag{2.25}$$

Adding this to equation (2.24) we see that together they simplify slightly:

$$(2.24) + (2.25) = np(1-p)(\mu_1 - \mu_0)(p + 1 - p) = np(1-p)(\mu_1 - \mu_0). \tag{2.26}$$

This can be inserted into equation (2.23) to obtain

$$r_{X,Y} \approx \frac{np(1-p)(\mu_1 - \mu_0)}{(n-1)s_x s_y}. \tag{2.27}$$

The only unknowns remaining are the sample standard deviations. We start with s_y. We know that an unbiased estimate of the variance is found by using the sample variance, s_y^2, when it is defined by

$$s_y^2 := \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2. \tag{2.28}$$

As above, our estimate for p is $\bar{y}$. With approximately $n(1-p)$ terms having $y = 0$ and np terms having $y = 1$ our estimate for the variance becomes

$$s_y^2 \approx \frac{1}{n-1}\left[n(1-p)(0-p)^2 + np(1-p)^2\right] = \frac{np(1-p)}{n-1}. \tag{2.29}$$

Recall that Bessel's correction - division by $n-1$ instead of n - is necessary as we must have the unbiased estimate of the variance for equation (2.20) to be accurate.

The variance s_x^2 is a bit more complicated. The vector $(x_1, x_2, \ldots, x_n)$ contains a sample from a mixed distribution with mixing parameter p. This means that, given two probability distribution functions $g_0(x)$ and $g_1(x)$, the vector contains a sample of observations from the distribution

$$f(x) = (1-p)g_0(x) + pg_1(x). \tag{2.30}$$

Here $g_0(x) = N(x; \mu_0, \sigma_0^2 = 1^2)$ and $g_1(x) = N(x; \mu_1, \sigma_1^2 = 1^2)$. The mean of this distribution is, as above, $\mu = (1-p)\mu_0 + p\mu_1$. An illustration of this can be seen in figure 2.1. Everitt and Hand (1981) found that by equating the observed second order moment

$$\frac{1}{n}\sum_{i=0}^{n}(x_i - \bar{x})^2,$$

to the theoretical moment

$$\int (x-\mu)^2 f(x)\mathrm{d}x,$$

we have that

$$\frac{1}{n}\sum_{i=0}^{n}(x_i - \bar{x})^2 \approx (1-p)[\sigma_0^2 + (\mu - \mu_0)^2] + p[\sigma_1^2 + (\mu - \mu_1)^2]. \tag{2.31}$$

For simplicity we will not delve into this derivation. Using $\mu = (1-p)\mu_0 + p\mu_1$, applying Bessel's correction and setting $\sigma_0 = \sigma_1 = 1$ we find that the sample variance can be expressed as

$$s_x^2 \approx \frac{n}{n-1}(1 + p(1-p)(\mu_1 - \mu_0)^2). \tag{2.32}$$

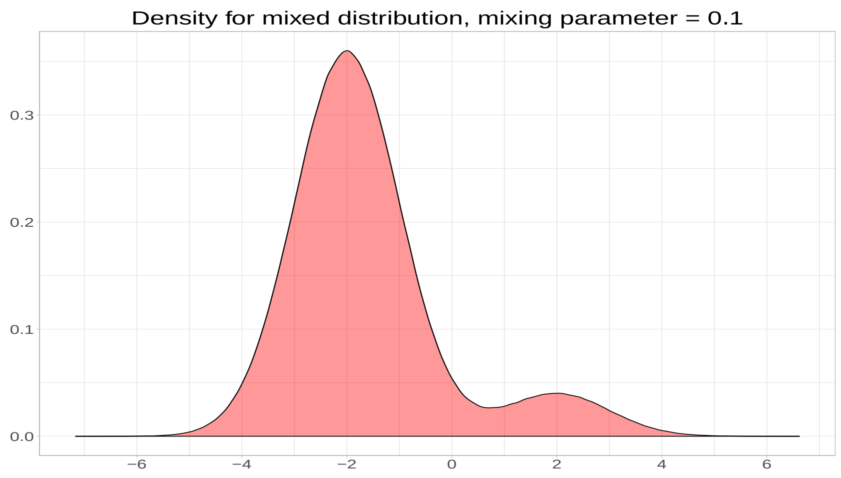


Figure 2.1: The density of the mixed distribution $f(x)$ with mixing parameter $p = 0.1$ as per equation (2.30). Here we have $g_0(x) = N(-2, 1^2)$ and $g_1(x) = N(2, 1^2)$.

Taking the square root and inserting equations (2.29) and (2.32) into equation (2.27) we obtain an expression for the sample correlation:

$$r_{X,Y} \approx (\mu_1 - \mu_0)\sqrt{\frac{p(1-p)}{1 + p(1-p)(\mu_1 - \mu_0)^2}}. \qquad (2.33)$$

Looking at this expression, we see that the magnitude of the correlation is determined by the distance $|\mu_1 - \mu_0|$ and by the imbalance in the data set.

From this we can clearly see that the correlation approaches zero as the imbalance in the data set increases. In the case of having a single explanatory variable one would generally wish to plot the response against the variable to better understand the relationship between the two. The more imbalanced the data set becomes, the harder it will be to visually determine a relationship. Research into a topic or field is often inspired by anecdotal evidence of some relationship between two or more variables, which is made significantly more apparent when variables are highly correlated.

2.2 Logistic Regression

Logistic regression is a very common method for performing classification. Closely resembling linear regression in many ways, it is a well understood tool and will here be used as the primary method for classifying data sets of varying imbalances. We will discuss the logistic model estimation of the coefficients, and briefly touch upon model selection.

2.2.1 The Logistic Model

We are to model the response Y_i for customer i given the data pairs

$$(y_i, \boldsymbol{x}_i)^T = (y_i, x_{i1}, x_{i2}, \dots, x_{ik})^T, \quad i = 1, 2, \dots, n,$$

where $\boldsymbol{x}_i$ is an observation vector containing k covariates with known response $Y_i = y_i$. We have n of these observations and the response is binary, with $Y_i \in \{0, 1\}$. We denote $p_i = P(Y_i = 1|\boldsymbol{x}_i)$ and we have that

$$Y_i|\boldsymbol{x_i} \sim \text{Bin}(1, p_i). \tag{2.34}$$

The expectation and variance of this distribution is

$$\mathrm{E}(Y_i) = p_i, \quad \mathrm{Var}(Y_i) = p_i(1 - p_i). \tag{2.35}$$

With heteroscedastic variance depending on the probability p_i we can use generalized linear models to model the response. Defining

$$\eta_i = \beta_0 + x_{i1}\beta_1 + \dots + x_{ik}\beta_k = \boldsymbol{x}_i^T\boldsymbol{\beta} \quad \text{and} \quad \boldsymbol{x}_i^T := (1, x_{i1}, x_{i2}, \dots, x_{ik}) \tag{2.36}$$

where η is our linear predictor and $\boldsymbol{\beta}$ is a vector of parameters yet to be determined, we have that for the binomial distribution the *canonical link function* g (Fahrmeir et al., 2013) is defined by

$$\eta_i = g(p_i) = \ln\left(\frac{p_i}{1 - p_i}\right). \tag{2.37}$$

We define h as its inverse so that $p_i = h(\eta_i)$. Solving the above for p_i we find that

$$p_i = h(\eta_i) = \frac{e^{\eta_i}}{1 + e^{\eta_i}}, \tag{2.38}$$

which is our response function. We can see that this function satisfies several important criteria:

$$h : \mathbb{R} \to (0, 1), \quad h'(\eta) > 0, \quad \lim_{\eta \to -\infty} h(\eta) = 0 \quad \text{and} \quad \lim_{\eta \to \infty} h(\eta) = 1. \tag{2.39}$$

A function satisfying these criteria can be considered to be a probability measure, therefore p can be viewed as a probability.

Inference on the coefficients $\beta_j, j = 0, 1, \dots, k$, is usually based on the odds ratio, which is commonly used in betting. The ratio

$$\frac{P(Y_i = 1|\boldsymbol{x}_i)}{P(Y_i = 0|\boldsymbol{x}_i)}$$

is the odds for the event $Y_i = 1$, and using equation (2.37) we see that the odds becomes

$$\frac{P(Y_i = 1|\boldsymbol{x}_i)}{P(Y_i = 0|\boldsymbol{x}_i)} = \frac{p_i}{1-p_i} = e^{\eta_i} = \exp(\beta_0) \cdot \exp(\beta_1 x_{i1}) \cdot \ldots \cdot \exp(\beta_k x_{ik}). \tag{2.40}$$

A unit increase in $\boldsymbol{x}_{ij}$ will serve as a multiplication of the odds by a factor $\exp(\beta_j)$, thereby increasing the odds if $\beta_j > 0$ and decreasing the odds if $\beta_j < 0$.

2.2.2 Estimating the Coefficients

For non-gaussian errors the coefficients should not be found using ordinary least squares. However, as with linear regression we can use maximum likelihood to obtain an estimate. Assuming that the responses Y_i are conditionally independent we can define the likelihood function L as

$$L(\boldsymbol{\beta}|y) = \prod_{i=1}^{n} f(y_i|\boldsymbol{\beta}) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}, \tag{2.41}$$

where f is the probability mass function of a binomial variable. Recall that $p = p(\boldsymbol{\beta})$, therefore the right hand side has $\boldsymbol{\beta}$ as a parameter. By defining $l = \ln L$ we can work with sums instead of products, and we find that

$$\begin{aligned} l(\boldsymbol{\beta}|y) &= \sum_{i=1}^{n} y_i \ln p_i + (1-y_i)\ln(1-p_i) = \sum_{i=1}^{n} y_i \ln\left(\frac{p_i}{1-p_i}\right) + \ln(1-p_i) \\ &= \sum_{i=1}^{n} y_i \boldsymbol{x}_i^T\boldsymbol{\beta} - \ln(1+e^{\boldsymbol{x}_i^T\boldsymbol{\beta}}), \end{aligned} \tag{2.42}$$

by using equations (2.36) and (2.37), and the fact that

$$\ln(1-p) = \ln\left(\frac{1+e^{\eta}}{1+e^{\eta}} - \frac{e^{\eta}}{1+e^{\eta}}\right) = \ln\left(\frac{1}{1+e^{\eta}}\right) = -\ln(1+e^{\eta}).$$

The score function $s(\boldsymbol{\beta})$ is defined as the derivative of l with respect to $\boldsymbol{\beta}$:

$$s(\boldsymbol{\beta}) := \frac{\partial l}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} y_i\boldsymbol{x}_i - p_i\boldsymbol{x}_i. \tag{2.43}$$

Setting this to 0 we should find a global maximum of this function, as we can see from equation (2.42) that the expression is concave, which ensures a unique, global maximum as long as none of the variables are linear combinations of each other. The desirable property of a concave log-likelihood function is ensured due to the fact that we have used the canonical link function for our generalized linear model, and our distribution is from the exponential family. [1]

In order to find the $\boldsymbol{\beta}$ that can make this expression equal 0, we must solve $k+1$ non-linear equations. Therefore, we need a numerical way of solving these equations. The commonly used function `glm` which is implemented in **R** uses the Fisher Scoring

[1]The binomial distribution is from the exponential family (Casella and Berger, 2002).

algorithm. It works by using a Taylor expansion around some starting point $\boldsymbol{\beta}^{(0)}$ and using an iterative method to find the optimal parameters. By defining the Fisher information $F(\boldsymbol{\beta})$ as

$$F(\boldsymbol{\beta}) := -\frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^T \partial \boldsymbol{\beta}} = \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T p_i (1 - p_i) \tag{2.44}$$

we can utilize its inverse to find the optimal values $\boldsymbol{\beta}$ by the following iteration:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + F^{-1}(\boldsymbol{\beta}^{(t)}) s(\boldsymbol{\beta}^{(t)}), \quad t = 0, 1, 2 \ldots \tag{2.45}$$

As long as F is of full rank, this should converge to the parameter vector that maximizes $L(\boldsymbol{\beta}|y)$.

2.2.3 Model Selection

There are many criteria that can provide some insight into how well a statistical model fits its data. Most of these are relative, i.e their magnitude alone is unimportant and their value should only be compared to that of the competing models. One such criteria to be minimized is the Bayesian Information Criterion, defined by

$$\text{BIC} = -2l + k \ln(n), \tag{2.46}$$

where l is the log-likelihood function defined above, and k is the total number of predictors included in the model. n is the size of the data set. The penalty $k \ln(n)$ seeks to reduce the number of covariates included in the model, thereby making it faster to fit to the data, less likely to overfit the data and more interpretable.

Another such criteria to be minimized is the AIC, which is of a similar form to the BIC, but with a less strict penalty term. One can use these two criteria to create subsets of the data to use for inference, where the one chosen by the BIC will be smaller than the one chosen by the AIC. Model selection is not a very important part in this thesis, so we will simply use the BIC in order to reduce the number of covariates and decrease the workload that goes into building the models.

2.3 Decision Trees & Random Forests

Logistic regression is often considered to be the fundamental building block for statisticians when performing classification. We now turn to methods based on decision trees, which are thought to mimic human decision making. We first describe how a decision tree is constructed, then we cover theory pertaining to random forests.

2.3.1 Constructing a Decision Tree

Decision trees can be used for regression, but their primary function is to perform classification. The simplest and most generic decision tree is visualized in figure 2.2, which

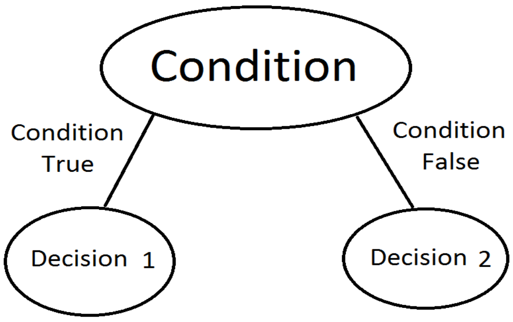


Figure 2.2: A generic decision tree with two terminal nodes.

makes decision 1 if the given condition is true. If it is false decision 2 is made instead. A simple example of this could be if we were to decide whether or not to go to the beach one day. We could condition this upon the weather the current day, say if it is sunny or not. If it is sunny we decide to go to the beach, and this would be decision 1. If it is not sunny the condition is false and we make decision 2, which is to stay at home. This process is quite easy and straightforward, and in many cases this is how decisions are made.

The decision tree in figure 2.2 has two terminal nodes, but in general decision trees can be much more complicated. The method of using trees for decision making relies upon the tree segmenting the predictor space R into J non-overlapping regions $R_j,\ j = 1, \dots, J$, and making the same prediction for each new observation that falls into a given region R_j. Note that the value of J depends upon the sample and is not given before the tree is made. As the regions are supposed to be disjoint, we should have $R_i \cap R_j = \emptyset, i \neq j$, and also $R = \cup_{j=1}^{J} R_j$ as these regions should make up the entire predictor space. In the case of k real valued continuous variables we have $R = \mathbb{R}^k$. Predictions are made by majority vote in each region, which means that when a new observation is to be classified, we check which region the observation falls into, and classify according to the rule given by equation (2.8). For a given cut-off probability α we classify the new observation as ω_1 if the proportion of observations in the training set within the region is higher than α, otherwise we classify the new observation as belonging to ω_0. A more detailed visualization of a decision tree is given in figure 2.3. In this figure we have the

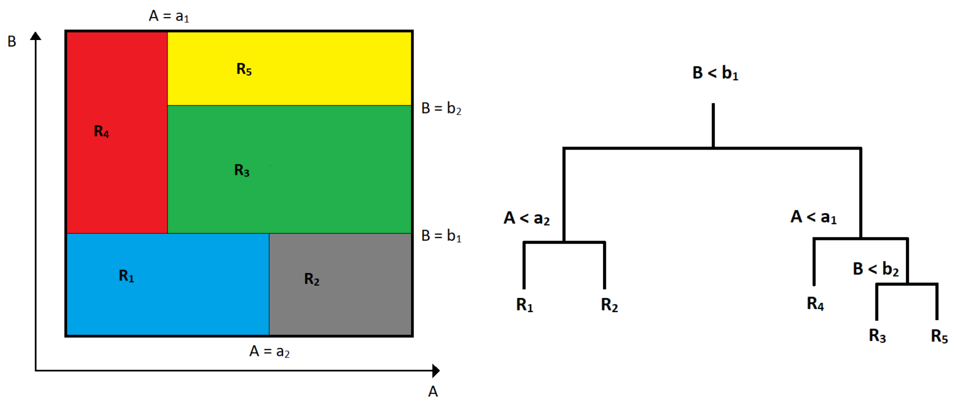


Figure 2.3: Left: A visual representation of the partition a decision tree with two predictors A and B can create. Right: The decision tree that partitions the predictor space as in the figure to the left. Splits in the decision tree have been made for A at a_1 and a_2 and for B at b_1 and b_2.

two predictors A and B that we can use to partition the predictor space. To the right we see the decision tree that created the boundaries visualized on the left. First, the condition $B < b_1$ was investigated. If a given condition is true we proceed down the left side of the split, otherwise we proceed to the right. Assume we have n_0 observations with label ω_0 and n_1 observations with label ω_1 in the data set. If we make no partition of the predictor space a new observation would have to be classified to the majority class, as this would be our guess. When the decision tree decides to investigate the condition $B < b_1$ it does so because on at least one side of $B = b_1$ the distribution of the class labels will be more uneven than in the data set as a whole. Assume that for $B < b_1$ we have M_0 and M_1 observations with respective labels ω_0 and ω_1. If the ratio M_0/M_1 is much bigger than 1 we can expect a new observation falling into this region to have the label ω_0. Similarly, if the ratio is much smaller than 1 we can expect a new observation to be of class ω_1. Decision trees in this way seek to find the boundaries that are capable of effectively discriminating between class labels. After having investigated the condition $B < b_1$ the decision tree continues by making the partition of the predictor space finer until the imbalance within a region is significant enough that the tree can accurately classify most observations within said region. Alternatively, the tree stops partitioning the predictor space when the number of observations in each region becomes sufficiently small. We note that for imbalanced data sets the decision tree might simply classify all observations to the majority class, as it is quickly satisfied with the distribution in each region and can accurately classify most observations. This further substantiates the claim by Rahman and Davis (2013) that most classification algorithms tend to work by maximizing accuracy.

Formally, when the splits are performed we are seeking the split that provides the greatest reduction in Gini Impurity $G(R_j)$ (Muchai and Odongo, 2014) for a given region

R_j, defined for C classes as

$$G(R_j) = \sum_{k=0}^{C-1} p_j(\omega_k)(1 - p_j(\omega_k)). \tag{2.47}$$

In our case $C = 2$. $p_j(\omega_k)$ is the probability of an observation in region R_j having class label ω_k and is simply the relative frequency of observations in R_j having this label. $1 - p_j(\omega_k)$ is the probability of misclassifying this observation. When a region R with n_R observations gets split into two smaller regions R_1 and R_2 with respective numbers of observations n_{R_1} and n_{R_2}, the Gini Impurity after this split has been performed becomes

$$G_{\text{Split}}(R) = \frac{n_{R_1}}{n_R} G(R_1) + \frac{n_{R_2}}{n_R} G(R_2), \tag{2.48}$$

and the split that is performed is the one that maximizes the difference $G(R) - G_{\text{Split}}(R)$.

2.3.2 Bootstrap Aggregating and Random Forests

Before we move onto random forests we must first look at bootstrap aggregating (bagging). When reading this keep in mind that random forests are essentially the same as bagged trees, with the only difference being which predictors we include when creating the individual trees.

Bootstrapping is a technique often used to approximate the sampling distribution of some statistic when the actual distribution is unknown. A drawback of decision trees is that they tend to be very dependent upon their training data, meaning that they are unstable when new observations are introduced and therefore have high variance. This variance can be reduced by making many trees and aggregating their predictions, thereby creating what is referred to as bagged trees. We know that if $\text{Var}(Y_i) = \sigma^2$, then $\text{Var}(\sum_{i=1}^{B} Y_i/B) = \sigma^2/B$ as long as the Y_i are independent and identically distributed.

Bootstrapping works by drawing a random sample with replacement M times from a data set of size M. We create a decision tree to this sample, and repeat this process B times. These trees are fitted to the same distribution and are not equal, but they are dependent upon each other. This means that if an observation $\boldsymbol{x}$ is classified by trees i and j with corresponding classifications Y_i and Y_j, then these are correlated with some positive correlation ρ. We have

$$\text{Cov}(Y_i, Y_j) = \begin{cases} \rho\sigma^2 & \text{if } i \neq j, \\ \sigma^2 & \text{if } i = j. \end{cases} \tag{2.49}$$

Because $\text{Var}(Y) = \text{Cov}(Y, Y)$, for the mean $\bar{Y} = \sum_{i=1}^{B} Y_i/B$ we have

$$\text{Var}(\bar{Y}) = \frac{1}{B^2} \text{Cov}\left(\sum_{i=1}^{B} Y_i, \sum_{j=1}^{B} Y_j \right). \tag{2.50}$$

There are B^2 terms here, with B of the terms having $i = j$ and covariance σ^2. The remaining $B^2 - B$ terms each have covariance $\rho\sigma^2$. This leads us to the variance for the mean

$$\mathrm{Var}(\bar{Y}) = \frac{1}{B^2}\left(B\sigma^2 + (B^2 - B)\sigma^2\rho\right) = \frac{\sigma^2}{B}(1 + \rho B - \rho). \tag{2.51}$$

We see that for $\rho = 1$ the variance is the same as for a single decision tree, and for $\rho = 0$ we have been able to divide the variance by B. The final prediction is then 1 if $\bar{Y} > \alpha$, with the cut-off α usually equal to 1/2. The cut-off can be set to any number, but must be determined before the random forest is created. In the limit $B \to \infty$ the variance approaches $\rho\sigma^2$. Typical values for B lie in the hundreds, therefore a good approximation is

$$\mathrm{Var}(\bar{Y}) \approx \rho\sigma^2. \tag{2.52}$$

The correlation ρ is necessarily less than 1, although how much smaller is difficult to say. Regardless, bagging can clearly improve the variance of the prediction.

We now move on to random forests. Random forests were created by Leo Breiman and Adele Cutler, and the implementation in **R** used here is based on their original code that was written in Fortran (Breiman, 2002). The idea is to further reduce the variance of the classifier by reducing the correlation ρ between the trees. While random forests - like bagged trees - train each tree on a random sample of the training set, random forests additionally draw a random sample with replacement of the predictors it can use at each split. When building a tree based on k predictors, it is recommended to use $m \approx \sqrt{k}$ at each split (Breiman, 2002). To clarify: these m predictors are drawn randomly several times when creating a tree, once for each split. The chosen predictor is the one that has the greatest reduction in Gini impurity, as for bagging. Trees grown on different samples with different sets of predictors must necessarily be even less correlated than bagged trees, leading to a further decrease in variance.

Another advantage of random forests is that they eliminate the need for validation sets to create an estimate of the test set error (Breiman, 2001). When M observations are drawn with replacement from a data set of size M, the probability of one of these observations not being drawn is $(1 - 1/M)^M$. We see that in the limit $M \to \infty$ this equals $1/e$ by definition, which means that approximately M/e observations are left out during the construction of a single tree. These observations are commonly referred to as the *out-of-bag*(OOB) sample and can be used as a validation set during training. The *out-of-bag error rate* is the error rate when applying the random forest to the current OOB sample during training, and this error rate is an unbiased estimate of the test set error rate, implying that the algorithm does not overfit the data (Breiman, 2001).

2.4 Sampled Data

The advantage of sampling our own data is the fact that we can decide exactly which conditions to examine. Training sets with imbalances of arbitrary degree can easily be created, and we can decide what kind of distribution our covariates are to follow. The normal (gaussian) distribution is central in many fields of statistics, and for this reason we will choose to look at explanatory variables that conditionally follow this distribution.

2.4.1 One Normally Distributed Covariate

Consider the case described in section 2.1.6, where we have one covariate X and a response Y and X follows a normal distribution, conditional upon the value of Y. Assume Z is a random variable following a standard normal distribution. Defining the function $\Phi(z)$ as the probability that $Z \leq z$, we then have that

$$\Phi(z) := P(Z \leq z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{1}{2}x^2} \mathrm{d}x. \tag{2.53}$$

Approximate values of this function can be found in a table, or one can use the **R** function `pnorm(z)`. This is commonly known to statisticians, and we can use this function to determine the TNR, TPR and BACC of a given logistic model. Recall that with one predictor $X = x$ the linear predictor is given by

$$\eta = \beta_0 + x\beta_x.$$

With a cut-off probability α of 0.5 we have that an observation will be classified as a positive instance when

$$p(Y = 1|\eta) > \frac{1}{2}, \implies \eta > 0, \implies x > -\frac{\beta_0}{\beta_x}, \tag{2.54}$$

given that $\beta_x > 0$. We can make sure that β_x is not a negative number from how we define the distribution of x, namely by having $\mu_1 > \mu_0$. Given that $Y = 0$, we have that

$$X - \mu_0 \sim Z \sim N(0, 1^2).$$

The true negative rate is the fraction of correctly classified negative observations. We can then see from equation (2.54) that this is given by

$$\begin{aligned} \text{TNR} &= P(X < -\beta_0/\beta_x) = P(Z + \mu_0 < -\beta_0/\beta_x) \\ &= P(Z < -\mu_0 - \beta_0/\beta_x) = \Phi(-\mu_0 - \beta_0/\beta_x). \end{aligned} \tag{2.55}$$

We note that $P(Z > z) = 1 - P(Z \leq z) = 1 - \Phi(z)$, and conditional upon $Y = 1$ we know that

$$X - \mu_1 \sim N(0, 1^2).$$

This positive observation will be correctly classified as long as $X = x > -\beta_0/\beta_x$. We then see that the true positive rate is given by

$$\begin{aligned} \text{TPR} &= P(X > -\beta_0/\beta_x) = P(Z > -\mu_1 - \beta_0/\beta_x) \\ &= 1 - P(Z < -\mu_1 - \beta_0/\beta_x) = 1 - \Phi(-\mu_1 - \beta_0/\beta_x). \end{aligned} \tag{2.56}$$

The balanced accuracy is then calculated by using the mean of these theoretical values. When a model has been fitted and values for β_0 and β_x have been obtained, we can calculate the TPR, TNR and BACC without having to rely on a test set. Obviously, this is only possible when we know the distributions the variables are coming from. The values should be more or less independent of initial seeds when we use sufficiently many observations. This naturally leads to the question: what happens when the number of available observations is small? This is a central question when dealing with imbalanced data, as sample sizes of an order of magnitude $n = 1000$ can in no way be considered large when dealing with the same imbalances as observed in the data set provided by Sparebank 1 (approximately 219 negative observations for each positive observation). We will be looking at the variability of the coefficients β_0 and β_x and their associated performance when fitted on small data sets in the Results section.

Chapter 3

Data Set provided by Sparebank 1 Kredittkort AS

3.1 Predicting Refinancing of Credit Card Debt

My project thesis (Frogner, 2019) concerned the problem of predicting which customers might seek refinancing of their credit card debt from competing banks. The data set was provided by Sparebank 1, and for this thesis a new data set has been provided. It consists of 614 465 observations of 78 variables, including response. A complete list of the variables with explanations can be found in Appendix A. The distribution of the response can be viewed in table 3.1. In the data set the response is named `ReFinInd`. It is worth nothing

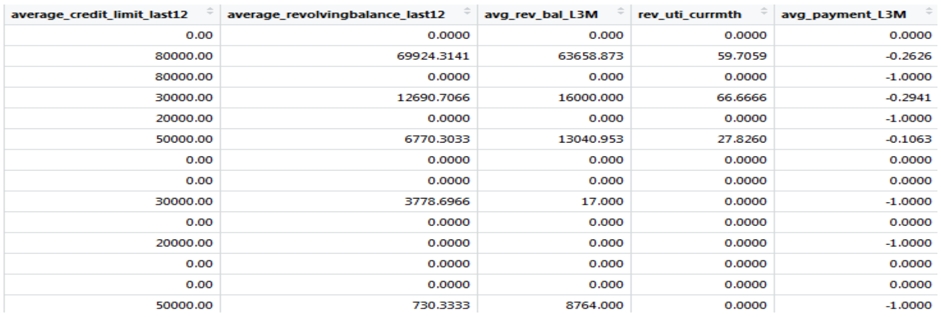
No Refinancing	Refinancing	Total
611 800	2665	614 465

Table 3.1: The number of customers who have received refinancing from competitors, as determined by Sparebank 1.

that the imbalance given in table 3.1 will change after we are done with data preparation, which is described in section 3.2. The data provided by Sparebank 1 will be used as a case study to compare the effectiveness of some of the methods covered in the theory section, as opposed to when the methods are applied to variables from ”nicer” distributions that we have sampled from.

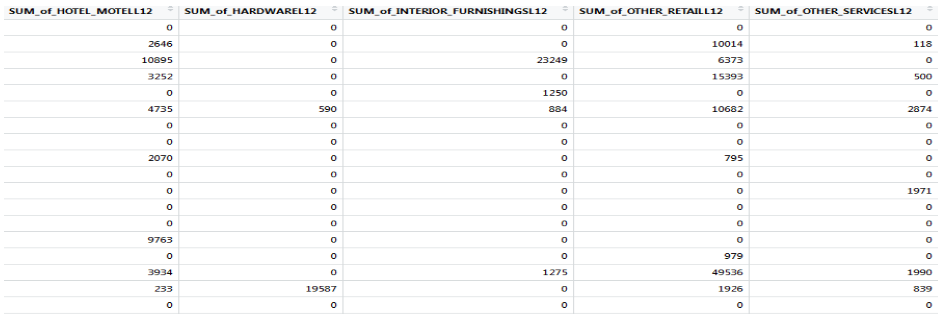
3.1.1 Distribution of the Covariates

We begin by looking at the data set to get some idea of what we are dealing with. Two excerpts of some of the variables can be seen in figures 3.1 and 3.2. These excerpts clearly illustrate that the distribution of the covariates do not follow a ”nice” distribution that you



average_credit_limit_last12	average_revolvingbalance_last12	avg_rev_bal_L3M	rev_uti_currmth	avg_payment_L3M
0.00	0.0000	0.000	0.0000	0.0000
80000.00	69924.3141	63658.873	59.7059	-0.2626
80000.00	0.0000	0.000	0.0000	-1.0000
30000.00	12690.7066	16000.000	66.6666	-0.2941
20000.00	0.0000	0.000	0.0000	-1.0000
50000.00	6770.3033	13040.953	27.8260	-0.1063
0.00	0.0000	0.000	0.0000	0.0000
0.00	0.0000	0.000	0.0000	0.0000
30000.00	3778.6966	17.000	0.0000	-1.0000
0.00	0.0000	0.000	0.0000	0.0000
20000.00	0.0000	0.000	0.0000	-1.0000
0.00	0.0000	0.000	0.0000	0.0000
0.00	0.0000	0.000	0.0000	0.0000
50000.00	730.3333	8764.000	0.0000	-1.0000

Figure 3.1: An excerpt of the data set. The variables regard credit limit, revolving balance, utilization and payments.



SUM_of_HOTEL_MOTELL12	SUM_of_HARDWAREL12	SUM_of_INTERIOR_FURNISHINGSL12	SUM_of_OTHER_RETAILL12	SUM_of_OTHER_SERVICESL12
0	0	0	0	0
2646	0	0	10014	118
10895	0	23249	6373	0
3252	0	0	15393	500
0	0	1250	0	0
4735	590	884	10682	2874
0	0	0	0	0
0	0	0	0	0
2070	0	0	795	0
0	0	0	0	0
0	0	0	0	1971
0	0	0	0	0
0	0	0	0	0
9763	0	0	0	0
0	0	0	979	0
3934	0	1275	49536	1990
233	19587	0	1926	839
0	0	0	0	0

Figure 3.2: An excerpt of the data set. The variables regard what categories the customers spend money on.

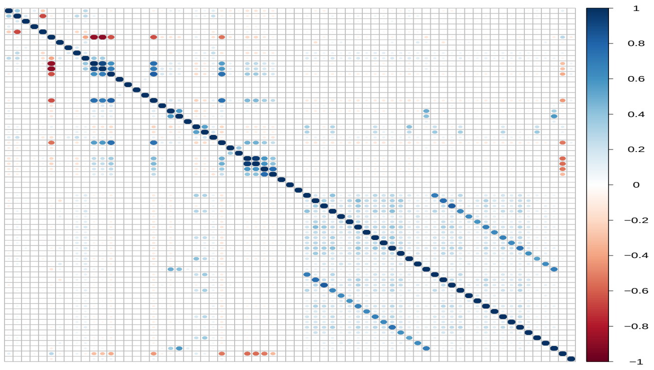


Figure 3.3: Correlation plot of the data set. Bottom row/last column is the response `ReFinInd`.

would find in a textbook in statistics. Many of the numbers are 0, and if they are non-zero then they might be numbers from a few hundred to tens of thousands. In fact, after performing the initial stages of data preparation there are 18.7 million 0-valued numbers split among 66 variables and 466 539 observations. This means that more than half of the numbers in the data set are 0.

A correlation plot of the data set is found in figure 3.3. The bottom row and last column show the correlation between the response and the other variables. As studied in section 2.1.6 we found that the correlation between a variable from a mixed normal distribution and a binomial variable will be quite low for a data set that is this imbalanced. Even though the variables here don't follow the same distribution, we see that the findings are much the same: the correlation between the variables and the response is very close to 0. Determining an accurate relationship between the variables and the response by inspection will be very difficult. This is obviously made more difficult for data sets with many variables.

3.2 Data Preparation

In this data set there are a few issues that must be dealt with before models can be fitted. These are mainly the issues of dealing with missing values, removing duplicate observations and handling multicollinearity between the variables.

Missing Values

There are 2 357 690 missing values in the data set. Upon inspection it is found that almost all these missing values are found in variables that are defined as some fraction of numbers. It is apparent that most - if not all - of these missing values are missing due to the fraction being 0/0. One example of this is the variable `AvgRevBalL3onL12` - average revolving balance last 3 months divided by average revolving balance last 12 months. Many customers have no revolving balance (they might not even use their credit cards), and in this case this variable will have an undefined value. In this case it makes sense to assume that this is the cause for the missing value, and it seems appropriate to define

$$\frac{0}{0} := 0$$

for the variables that might have this issue. The only variable that has missing values that can not have arisen from this issue is the variable `CustomerAge`. This variable has 39 missing values, and these observations were simply removed from the data set.

Duplicate Observations

The variable `BK_ACCOUNT_ID` is the unique anonymized account identifier. Sorting the data set according to this variable and checking to see if any two consecutive observations share the same account identifier we can identify duplicate observations. A total of 3700 duplicate observations were identified and removed. In conversations with Sparebank 1 we have discussed the methods of SMOTE and oversampling, as per section 2.1.4. It is believed that some of these duplicates might be due to oversampling performed by Sparebank 1, although exactly how the duplicates have arisen is unknown; they could also be due to technical errors. Nevertheless, these observations were removed in order to maintain control of the rate of oversampling and to maintain the purity of the data set.

Removing Unwanted Data

In the theory section several methods for making the data set more balanced were discussed. These methods were undersampling, oversampling and SMOTE. We can also simply look at the variables and see if any of these can uniquely determine the response. The variable `Segment9Name` is a variable that segments the customer base according to some criteria. One of the levels of this categorical variable is `"Not active in last 12 mths"`. Of the 144 199 accounts with this level, only 9 accounts had a positive response `ReFinInd`; a fraction of less than 1 in 16 000. This is the kind of variable that is easy to notice through inspecting the data: an account that has not been active for 12 months is very unlikely to receive refinancing from a competitor. As the vast majority of these observations are negative and we are interested in identifying positive observations, we choose to omit these observations. This favorably changes the distribution of the response by making the data set more balanced, and the final distribution that we end up with can be seen in table 3.2. The imbalance in this data set is then roughly $p = 1/220$;

No Refinancing	Refinancing	Total
464 419	2120	466 539

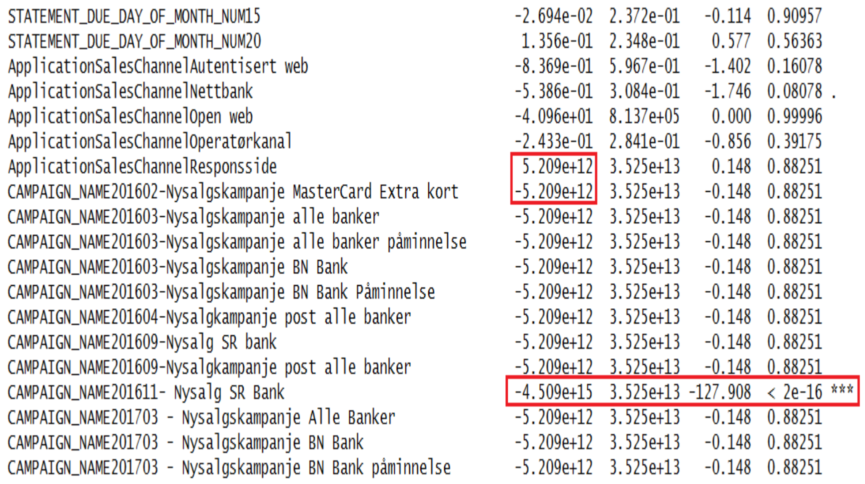
Table 3.2: The distribution of the response in the data set after preparing the data as described in this section.

219 negative observations for every positive observation. This value for p will be used extensively in the Results section. The data set we will be working with will consist of 466 539 observations, before splitting into training and test set. Additional variables will be removed from the data set, but from here on out additional observations will not be removed, except when performing undersampling.

Another issue that must be dealt with is that of categorical variables having many levels. For instance, the variable `CAMPAIGN_NAME` has 81 factor levels. Breiman and Cutler's implementation of random forests originally required categorical variables to have fewer than 33 values (Breiman, 2002). Determining the optimal splits in a categorical variable to create a decision tree is a problem that grows exponentially as a function of the number of factor levels. For a variable with 81 levels this can take an extreme amount of time, as we note that $2^{80} > 10^{24}$. No modern computer can deal with computations of this magnitude in any reasonable amount of time. Variables with many levels have also been identified as problematic when testing a logistic model trained on an undersampled data set on new data.

Multicollinearity

The final issue that must be dealt with is that of collinearity between the variables. Consider figure 3.4. This figure shows part of a summary from fitting a logistic model to the



```
STATEMENT_DUE_DAY_OF_MONTH_NUM15                              -2.694e-02  2.372e-01   -0.114  0.90957
STATEMENT_DUE_DAY_OF_MONTH_NUM20                               1.356e-01  2.348e-01    0.577  0.56363
ApplicationSalesChannelAutentisert web                        -8.369e-01  5.967e-01   -1.402  0.16078
ApplicationSalesChannelNettbank                               -5.386e-01  3.084e-01   -1.746  0.08078 .
ApplicationSalesChannelOpen web                               -4.096e+01  8.137e+05    0.000  0.99996
ApplicationSalesChannelOperatørkanal                          -2.433e-01  2.841e-01   -0.856  0.39175
ApplicationSalesChannelResponsside                             5.209e+12  3.525e+13    0.148  0.88251
CAMPAIGN_NAME201602-Nysalgskampanje MasterCard Extra kort     -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201603-Nysalgskampanje alle banker               -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201603-Nysalgskampanje alle banker påminnelse    -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201603-Nysalgskampanje BN Bank                   -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201603-Nysalgskampanje BN Bank Påminnelse        -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201604-Nysalgkampanje post alle banker           -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201609-Nysalg SR bank                            -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201609-Nysalgkampanje post alle banker           -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201611- Nysalg SR Bank                           -4.509e+15  3.525e+13 -127.908  < 2e-16 ***
CAMPAIGN_NAME201703 - Nysalgskampanje Alle Banker             -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201703 - Nysalgskampanje BN Bank                 -5.209e+12  3.525e+13   -0.148  0.88251
CAMPAIGN_NAME201703 - Nysalgskampanje BN Bank påminnelse      -5.209e+12  3.525e+13   -0.148  0.88251
```

Figure 3.4: Summary of a logistic model, displaying issues of multicollinearity between some variables.

data set. Which variables are included can be found in Appendix B, under the print-out of the aptly named model `modelUseless`. The first red box clearly displays collinearity between variables; we have two coefficients that are extremely large, with the same order of magnitude and opposite signs. In fact, almost all levels of `CAMPAIGN_NAME` share the same coefficient. This inflation of the coefficients occurs due to a near-singular design matrix, and we can see from the right-most column that the p-values are equal to 0.88 which typically means that this is a value to be discarded. However, a few rows down we find one factor level that has a coefficient several orders of magnitude greater than the others, and this coefficient is actually determined to be significant by looking at the associated p-value. It would clearly be a mistake to include this variable. Recall that inference on the parameters is usually based on the odds ratio, as given in equation (2.40). This means that - given two observations $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ that only differ in this variable level - the odds relative to each other will be equal to

$$\frac{\frac{p_1}{1-p_1}}{\frac{p_2}{1-p_2}} = \exp(4.509 \cdot 10^{15}).$$

One can verify that this number will have more than 10^{15} digits, making it absurdly large. While it was not entirely necessary to show just how large this number is, it was interesting to point out exactly how poor the results from logistic regression can be when dealing with collinear variables. On the other hand, other methods will have no issue in dealing with collinear variables. Random forests is one such example.

For the reasons stated, the variable `CAMPAIGN_NAME` was removed from further inference on the data set. The variable `Segment9Name` is explicitly given by the variable `Segment23Name`, and the latter was removed from the data set. Several variables were removed, as they all contained the same information, namely the period of time the data set was taken from.

Chapter 4

Results

4.1 About the Results

The results have been produced using the statistical software **R**. Results that have been produced by sampling data are entirely reproducible, and the reader can reproduce any of these results by running the code given in Appendix C. The results pertaining to the data set provided by Sparebank 1 are unfortunately not reproducible, as the data set cannot be made available.

Regarding notation: notation in the results section will as closely as possible follow the same notation used in the theory section. When comparing models to each other, we need to be able to differentiate between them. What separates one model from another is often no more than the data set the model is trained on, which can be oversampled, undersampled or SMOTEd. The name of the model will reflect this in some way, and will be explained as thoroughly as possible when necessary. This is of higher importance when working on the data set provided by Sparebank 1. We denote by p the imbalance in the original data set we are using, and p_t is the imbalance in the resampled training set, as per section 2.1.5. The degree of resampling will sometimes be referred to as p_t, as these are one and the same.

4.2 Logistic Regression on Sampled Data

In this section we will be using logistic regression on sampled data of varying levels of imbalance to investigate the effectiveness of the methods discussed in the Theory section.

4.2.1 One Normally Distributed Predictor

We begin by studying the case described in section 2.1.6. We have a response Y and a covariate X, where

$$\begin{aligned} &Y \sim \text{Bin}(1, p), \quad p \in [1/1000, 1/2], \\ &X|(Y=0) \sim N(-1, 1^2), \\ &X|(Y=1) \sim N(1, 1^2). \end{aligned}$$

Until otherwise stated, this will be the case we are studying. We vary p between 1/2 - the perfectly balanced case - and 1/1000, which is quite an imbalanced data set. All parameters are considered to be known, we therefore do not need to estimate any of these parameters. We have sampled 1000 points from the minority class, where the distribution is as above. We also sample the appropriate number of points from the majority class to obtain the specified imbalance in the data set. For instance, in the case of an imbalance of 1/1000 we have sampled 1000 points from the minority distribution and 999 000 from the majority distribution. A logistic model was fitted to each of these data sets and the results can be viewed in table 4.1. The cut-off probability α here is 1/2, which means that the linear

$P(Y=1)$	β_0	β_x	$-\beta_0/\beta_x$	TNR	TPR	BACC	Accuracy
1/2	0.098	1.99	-0.049	0.83	0.85	0.84	0.841
1/5	-1.34	2.05	0.65	0.95	0.64	0.79	0.885
1/10	-2.14	1.98	1.08	0.98	0.47	0.72	0.931
1/20	-2.95	2.01	1.47	0.993	0.32	0.66	0.960
1/50	-3.86	1.97	1.96	0.998	0.17	0.58	0.982
1/100	-4.58	1.97	2.32	0.999	0.09	0.55	0.991
1/220	-5.47	2.09	2.62	0.999	0.05	0.53	0.996
1/500	-6.17	1.96	3.15	0.999	0.016	0.51	0.998
1/1000	-6.90	1.99	3.47	0.999	0.007	0.50	0.999

Table 4.1: Performance of logistic regression on data sets of varying imbalances. The value in red corresponds to the imbalance of the data set from Sparebank 1.

predictor $\eta_i = \beta_0 + x_i\beta_x$ must be greater than 0 for Y_i to be classified as positive. This is ensured whenever $x_i > -\beta_0/\beta_x$ and this discriminating value is shown in the fourth column of the table. We note that in the perfectly balanced case we have

$$\text{E(TNR)} = \text{E(TPR)} = \text{E(BACC)} = \text{E(Accuracy)} = \Phi(1) \approx 0.841,$$

where $\Phi(z)$ is as described in equation (2.53). Simple reasoning for this is the fact that a true positive or true negative should be equally likely in this case, and will occur at a rate of $\Phi(1)$. The coefficients β_0 and β_x are also included in the table, and we see that for higher degrees of imbalance the intercept β_0 steadily decreases, while the coefficient β_x seems to remain stable around 2. This causes the discriminant $-\beta_0/\beta_x$ to increase, which means that increasing degrees of imbalance are more likely to classify a new observation as negative, regardless of which class it belongs to. The TNR, TPR and BACC given here are not based on a test set, they are calculated using theoretical values, as per equations

(2.55) and (2.56). By theoretical values we mean theoretical as long as β_0 and β_x are given. In the way this small experiment is set up, we know that we have a variable that has significant explanatory power. In the perfectly balanced case we have a balanced accuracy of 0.84, with the true negative and true positive rates more or less the same. When the imbalance becomes $p = 1/10$ the true positive rate has fallen below 50 %, meaning that we are no longer able to correctly classify even half the positive observations, given this model. We see that as p decreases below 1/50 the true negative rate has more or less reached 1, while the true positive rate continues to decrease. At this point we appear to be reaching the threshold for where logistic regression can accurately identify any of the observations from the minority class, even though we have a variable that should be a strong indicator of which class the response belongs to.

Varying Cut-Off Probability

We now turn to the same case of using logistic regression to classify data sets of varying imbalance, but now with varying cut off probability α. A natural question is what kind of values should we use for α. As we saw from table 4.1 the BACC declines as the distribution becomes more imbalanced. This is due to the true positive rate decreasing faster than the true negative rate can increase. By adjusting down the cut-off probability we can increase the balanced accuracy, and we choose to investigate the same values for α as we have used for the imbalance p. The results of fitting a logistic model to data sets of varying imbalances with varying cut-off values can be see in table 4.2. The balanced

α/p	1/2	1/5	1/10	1/20	1/50	1/100	1/220	1/500	1/1000
1/2	0.846	0.786	0.714	0.653	0.591	0.534	0.519	0.506	0.503
1/5	0.789	0.835	0.825	0.783	0.698	0.624	0.573	0.544	0.515
1/10	0.722	0.812	0.851	0.831	0.766	0.690	0.627	0.576	0.536
1/20	0.648	0.769	0.827	0.850	0.824	0.758	0.692	0.620	0.577
1/50	0.577	0.692	0.762	0.817	0.845	0.830	0.767	0.699	0.649
1/100	0.539	0.631	0.701	0.766	0.828	0.841	0.820	0.756	0.702
1/220	0.514	0.576	0.632	0.694	0.774	0.819	0.842	0.809	0.769
1/500	0.507	0.536	0.575	0.623	0.704	0.762	0.821	0.845	0.814
1/1000	0.502	0.518	0.540	0.578	0.643	0.696	0.776	0.833	0.838

Table 4.2: Balanced Accuracy for logistic models trained on data sets of varying imbalances p when the cut-off probability α is varied.

accuracy for each case is given, with columns corresponding to a given imbalance p and rows corresponding to a given cut-off α. The highest value for each data set with imbalance p is highlighted in red. All these values appear to lie on the diagonal, where $p = \alpha$. The red values are clearly similar to the theoretical value $\Phi(1) \approx 0.841$ from equation (2.53). Above the diagonal we have that the true negative rate is high and the true positive rate is low, while below the diagonal the opposite is true. Viewing this as a matrix, we have that element (α, p) is approximately equal to (p, α). A BACC of 0.5 is the expected performance of random guessing, and we can see that several values are close to this value, which means that for some of these parameters the model is more or less entirely useless. This is a clear indication that varying the cut-off probability can improve a model significantly. With the BACC as our performance measure, a logistic model trained on a data set with imbalance $p = 1/1000$ can go from being about as effective as random guessing when using the usual cut-off $\alpha = 0.5$, to achieving a BACC as high as a model trained on a perfectly balanced data set when α is set equal to p.

4.2.2 Resampling techniques on small data sets

We now delve into the problem of examining the performance of the resampling techniques undersampling, oversampling and SMOTE. Here we will be looking at data sets with few positive observations, and we will compare our findings to those in the next section where we look at bigger data sets.

Several times in this section we will be writing ”parameters produced by oversampling”, or similarly. By writing this we are referring to the parameters β_0 and β_x of a logistic model trained on a resampled data set. This is to clarify that the simple act of resampling does not alone produce any specific parameters; the parameters will depend on the method of resampling as well as the degree of resampling, the random seed used for sampling and on the data itself.

Denoting the number of positive observations as n_1, we set $n_1 = 20$ and use the same value for p as in the data set from Sparebank 1, namely 1/220. This gives us 20 observations with label 1 from a $N(1, 1^2)$-distribution, and 4380 observations with label 0 from a $N(-1, 1^2)$-distribution, for a total of 4400 observations. We create 10 data sets following this distribution, then proceed to train a total of 30 models on these after having used all three resampling techniques to obtain training sets with an even distribution of positive to negative observations. For the undersampled training set we then end up with 20 positive and 20 negative observations, while we need an additional 4360 observations of the positive class for our oversampled and SMOTEd training sets. Fitting logistic models to each of these 30 sets will yield slightly different parameters (β_0, β_x) each time, and we can see a scatter plot of these in figure 4.1. In the top right we find a significant outlier. On

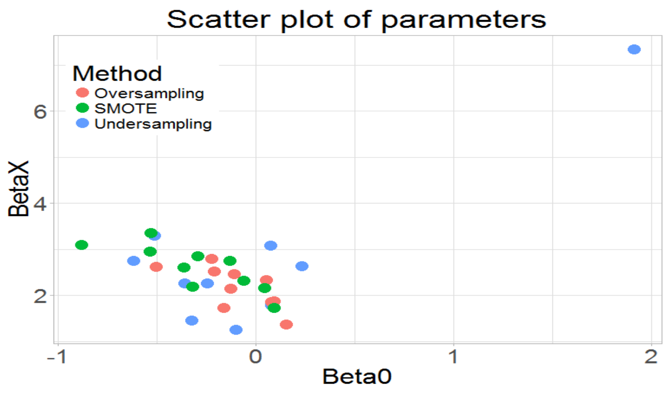


Figure 4.1: Scatter plot of parameters from logistic models trained on training sets with $p_t = 1/2$, original data with $p = 1/220$ and 20 positive observations.

the 6th data set that was sampled, three logistic models were trained, as for the other data sets. The parameter vectors for the models trained on this training set were $(1.91, 7.35)$, $(-0.21, 2.52)$ and $(-0.53, 2.95)$ when the resampling techniques were undersampling, oversampling and SMOTE, respectively. The parameters produced by using oversampling and SMOTE are nothing out of the ordinary and fit well with the cluster in the bottom left. Unsurprisingly, it was the undersampled training set that produced the outlier. Estimation of the parameters depends upon the data provided, and with small data sets the parameters are more likely to be influenced by a few noisy observations. Ignoring the outlier, it still seems that the parameters produced by undersampling the data have higher variability than those where the data has been oversampled. The parameters produced by SMOTE seem to have slightly higher variability than the ones produced by oversampling, but still less than the ones given by undersampling. We consider this stability to be a desirable property.

Twenty of the parameter vectors have an intercept β_0 that is negative, while the remaining 10 have positive intercepts, making a false negative more probable than a false positive. It is entirely possible that this is due to chance, as a sample size of 30 models is quite small. Measuring performance again by BACC, we can see the performance of the models in table 4.3. In this table we have also included a logistic model trained on

Run	Undersampling	Oversampling	SMOTE	No Resampling
1	0.843	0.843	0.843	0.508
2	0.840	0.843	0.839	0.552
3	0.839	0.842	0.841	0.526
4	0.842	0.842	0.842	0.516
5	0.844	0.843	0.842	0.529
6	0.836	0.843	0.839	0.524
7	0.837	0.838	0.834	0.528
8	0.842	0.842	0.842	0.520
9	0.837	0.842	0.840	0.526
10	0.843	0.843	0.842	0.517
Mean	0.840	0.842	0.840	0.525

Table 4.3: BACC for each test run for each of the models. The model with outlying parameters is marked in red. The test set consisted of 10 000 positive observations and 10 000 negative observations.

the original, imbalanced data set. All cut-off values α are 0.5. The BACC of the model that produced the outlier in figure 4.1 is highlighted in red, and is one of the lowest values in the table, ignoring the last column. The BACC of this model is still quite high, which shows that very different parameters can yield more or less the same results. Even though the 10 data sets were sampled from exactly the same distribution, we can see that the 7th run yields worse results than all the other runs. This is something that can happen when the models are trained on a particularly noisy training set, which seems to be the issue here. All the models were tested on the same test set, with observations drawn according to the case described at the beginning of section 4.2.1, with $p = 1/2$. Recall that the highest expected accuracy and BACC of a model tested on this distribution is $\Phi(1) \approx 0.8413$. Oversampling appears to achieve a mean BACC of 0.842, slightly higher than the theoretical threshold, and this must be due to chance. Undersampling and SMOTE both get an average of 0.840, very close to the theoretical threshold. It appears that for such a simple data set, undersampling can perform more or less just as well SMOTE and oversampling, even though the sample size is very small.

4.2.3 Resampling Techniques on Large Data Sets

Undersampling was shown to have higher variability than the other resampling techniques when estimating the parameters of a logistic model, but appeared to achieve roughly the same accuracy and BACC. Setting the number of positive observations n_1 equal to 1000, we investigate what happens to the parameters as we attempt to perform the same experiment on a bigger data set. We resample the training sets so that $p_t = 1/2$, and in figure 4.2 we see the 30 parameter pairs (β_0, β_x) that are produced from these models. In

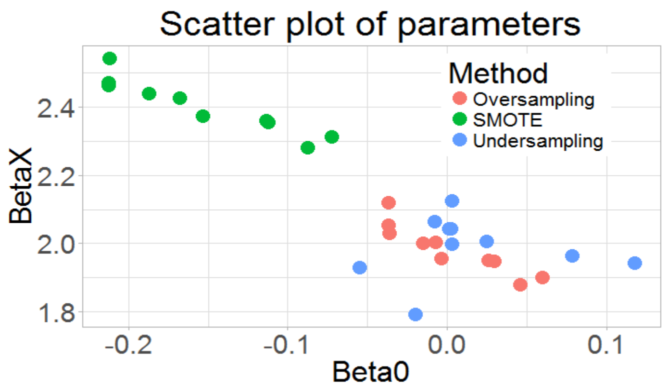


Figure 4.2: Scatter plot of parameters from logistic models trained on training sets with $p_t = 1/2$, original data with $p = 1/220$ and 1000 positive observations.

this second scatter plot we can see something interesting that was not apparent in the first one. The parameters produced by undersampling and oversampling appear to be clustered together, while the ones produced by SMOTE are clearly separated from the rest. Additionally, all the intercepts given by SMOTE are negative, and the mean true negative rate for SMOTE was 0.858, while the mean true positive rate was 0.826. We expect a fraction of $1 - \Phi(1) \approx 0.159$ positive observations to have the explanatory variable $X < 0$. When running SMOTE on data sampled from a $N(1, 1^2)$ distribution, we end up with a fraction of roughly 0.115 of the new data points having $X < 0$. Recall that whenever we use SMOTE in this thesis, the mixing parameter a is sampled uniformly from the interval [0.5,1], which means that this result will not be the same for other implementations of SMOTE. We can easily determine that both oversampling and undersampling will asymptotically produce data sets where the positive observations follow a $N(1, 1^2)$ distribution. SMOTE has yielded similar results as the other resampling techniques thus far, but we can see that the data it produces is not equal in distribution to the input data.

4.3 Data from Sparebank 1 Kredittkort AS

We now turn to the data set graciously provided by Sparebank 1 Kredittkort AS. We will apply random forests to the data set and compare their performance to that of logistic regression, and we will be using the resampling methods more extensively here.

4.3.1 Logistic Regression without Resampling

First, we partition the data set into a training set and a test set, using stratified sampling. The training set consists of 316 539 observations, with 1438 of these being positive observations. Similarly, the test set consists of 150 000 observations with 682 of these being positive observations. Including the response `ReFinInd`, the data set consists of 66 variables after we are done with data preparation. We fit a logistic model to the training set, and it is tested on the test set for different values of the cut-off probability α. This can be seen in table 4.4, and we have utilized all the possible variables, not counting the ones removed during data preparation. We term this model the "full model", and exactly which variables go into it can be seen in Appendix 6.2. Here we can clearly see the issues of an

α	TN	FP	FN	TP	BACC	Accuracy
1/2	149317	1	682	0	0.500	0.995
1/5	149312	6	682	0	0.500	0.995
1/10	149246	72	679	3	0.502	0.995
1/20	148581	737	648	34	0.522	0.991
1/50	141110	8208	467	215	0.630	0.942
1/100	127864	21454	246	436	0.748	0.855
1/220	111391	37927	106	576	0.795	0.746
1/500	87835	61483	37	645	0.767	0.590
1/1000	77079	72239	25	657	0.740	0.518

Table 4.4: Performance of logistic regression on the data set given by Sparebank 1, utilizing all variables.

imbalanced data set: given the traditional cut-off value of 1/2, only a single observation has been classified as positive, which turns out to be a false positive. The balanced accuracy is equal to the expected value of random guessing, namely 0.5, while the accuracy is quite high. Fortunately, it is easy to adjust α for a logistic model, and we can see that the BACC increases significantly as α is adjusted down. This comes at the cost of accuracy, and for Sparebank 1 this trade-off is important. Their goal is to limit the monetary loss they have from customers seeking refinancing from competitors, but there will also be a loss associated with the false positives that increase rapidly in number as α decreases. When α is set to 0.01 we can see that the true positive rate has almost reached 2/3, and the BACC has climbed to almost 3/4. Adjusting α further down so that it equals the imbalance of $p = 1/220$ we find that the BACC is maximized among these chosen values for α. In section 4.2.1 we found that the optimal cut-off for data sets drawn from a mixed normal distribution seemed to be equal to the imbalance of the data set, and the same appears to be true here. By optimal, we of course mean optimal in conjunction with using BACC as our performance measure.

4.3.2 Model Selection

Model selection is typically performed in order to make a model more interpretable and less likely to overfit the data. We are not particularly interested in having an interpretable model, as the focus here is not understanding which variable to use, but rather which methods are more effective when classifying imbalanced data. However, there is a third reason why we would want to reduce the number of covariates included in the model. In section 4.3.4 we will fit several logistic models to the data set, after having used resampling techniques. The biggest of these required 870 MB of memory, and we will be looking at several models simultaneously. Dealing with several such models on a laptop can be an arduous task. Random forests also require significant amounts of memory, in addition to taking some time to fit. Random forests typically require hundreds, sometimes even thousands of decision trees in order to converge (Frogner, 2019). See the figures in section 4.3.3 for what it means for a random forest to converge. Reducing the number of covariates in the model can be a significant help in dealing with these issues, and in figure 4.3 we see the BIC for different model complexities. The covariates are chosen by forward selection,

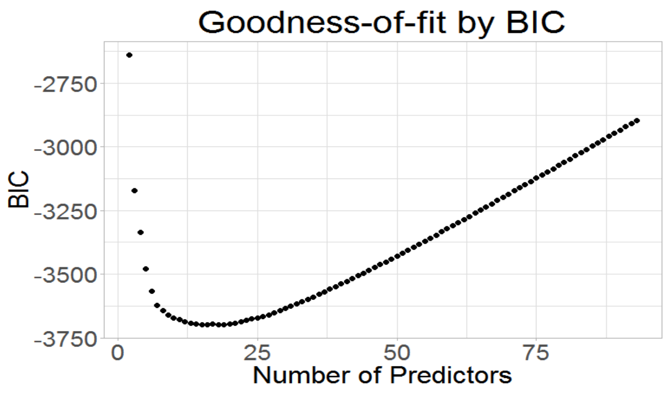


Figure 4.3: BIC by forward selection. A model with 17 variables is considered optimal by this criterion.

and a model with 17 predictors is deemed optimal. Several of the predictors are in fact different factor levels of the same variable, which means that the model actually only includes 12 different variables. Including all factor levels this turns out be 33 parameters that must be estimated for a logistic model, including intercept. Exactly which parameters this is can be seen from the print-out in Appendix 6.3. Thirty three parameters to estimate is still a significant reduction from the 93 that were needed if we were to use the full model.

In table 4.5 we can see the performance of the reduced logistic model when applied to the test set. The only difference between this table and table 4.4 are the variables that were used to produce them. The highest BACC obtained here is attained when $\alpha = p$, which is as we have come to expect. Here, the highest BACC is 0.798, compared to 0.795 for the full model. While this might imply that the reduced model might be slightly

α	TN	FP	FN	TP	BACC	Accuracy
1/2	149318	0	682	0	0.500	0.995
1/5	149316	2	682	0	0.500	0.995
1/10	149253	65	679	3	0.502	0.995
1/20	148551	767	651	31	0.520	0.991
1/50	141886	7432	481	201	0.622	0.947
1/100	128618	20700	244	438	0.752	0.860
1/220	108615	40703	90	592	0.798	0.728
1/500	83866	65452	36	646	0.754	0.563
1/1000	76618	72700	26	656	0.737	0.515

Table 4.5: Performance of the reduced logistic model on the test set.

better, it is worth noticing that the accuracy for the full model is higher than the accuracy for the reduced model for the same value of α. A side-by-side comparison of the tables will show that the models follow each other closely in both accuracy, balanced accuracy and true positive rate, implying that the models are of similar quality when it comes to classification. However, the reduced model is clearly superior to the full model regarding the issues discussed above in this section.

4.3.3 Random Forests without Resampling

We now turn to random forests. Random forests have many hyperparameters to be set before they are trained, the most important being the number of predictors `mtry` tried at each split, and the number of trees to be grown `ntree`. All hyperparameters excluding the cut-off probability are set to default values, and `ntree` is set to 200 for all the results produced here.

Training Random Forests on Imbalanced Data

As we have seen in previous sections, the cut-off probability α must be lowered below the typical threshold of 1/2 in order for logistic regression to be able to accurately classify the positive observations. We expect the same to apply to random forests to some extent, which means that we must attempt to choose meaningful values of α. While logistic models can be modified after training by adjusting α, random forests must have this parameter set before the model is built.

Consider figure 4.4. Here we have built four random forests on the full data set, with differing values for α. We see that the algorithm has not converged even after 200 decision trees have been made when the values for α are 1/50 and 1/220. For the higher values 1/10 and 1/2 the algorithm seems to converge much more quickly. We note from the purple curve where $\alpha = 1/50$ that the error rate seems to oscillate, and this is even more apparent in figure 4.5, which was taken directly from my project thesis Frogner (2019). Higher values of α make the out-of-bag error rate converge more quickly, and the error rate is higher when α is small, as expected. We can see that for $\alpha = 0.05$ there is a clear oscillation of the error rate for the first 200 decision trees created, then it seems to stabilize

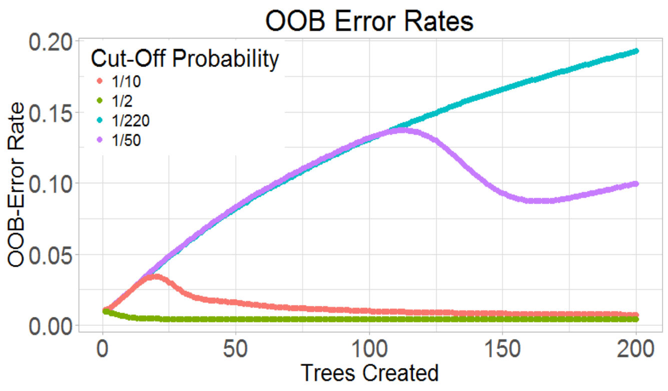


Figure 4.4: Out-of-bag error rate for random forests fitted on the full data set.

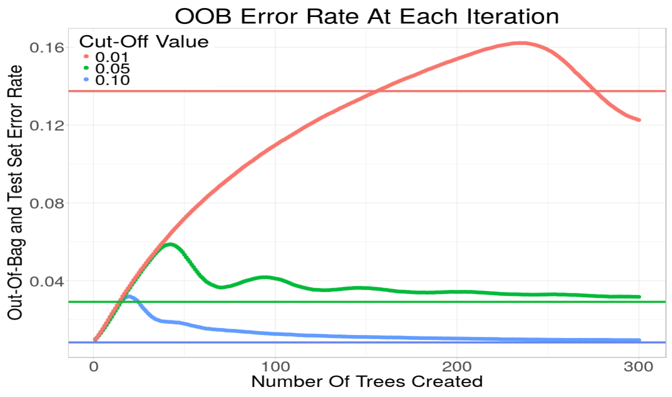


Figure 4.5: Oscillation of out-of-bag error rate. Taken directly from Frogner (2019).

towards the test set error rate.

The increase in the OOB error rate for lower values of α is due to the fact that these classifiers are able to achieve a higher true positive rate than the classifiers with higher values for α, at the cost of additional false positives. This is apparent in figure 4.6, where we can see the true positive rates on the out-of-bag samples during training. The shape of these graphs are more or less identical to the ones in 4.4. Combining this with the findings in figure 4.7 where we see the true negative rates for the four random forests, we find that adjusting α down has the same effect as for logistic models, namely increasing the BACC of the classifier. Adjusting α is necessary when studying imbalanced data sets, and unfortunately leads to the random forest requiring many trees in order to converge. This can be time consuming, and many computers will have memory issues as the number of trees in the forest becomes large. There are methods to combat these issues, for instance by creating several smaller forests and combining them after they have been trained.

Similar graphs were created for random forests fitted to the reduced data set, using the same hyperparameters that produced the results we have seen in figures 4.4, 4.6 and

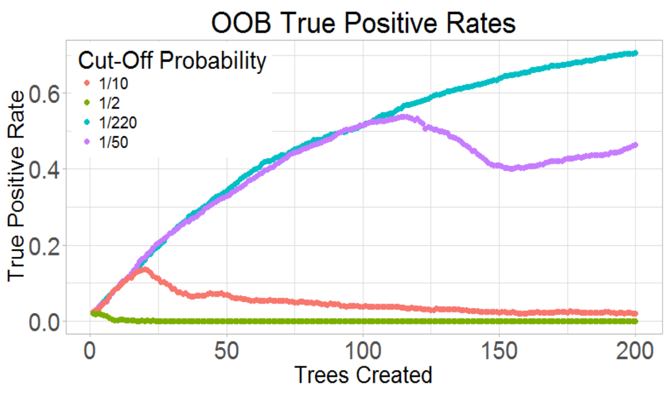


Figure 4.6: Out-of-bag true positive rate for random forests fitted to the full data set.

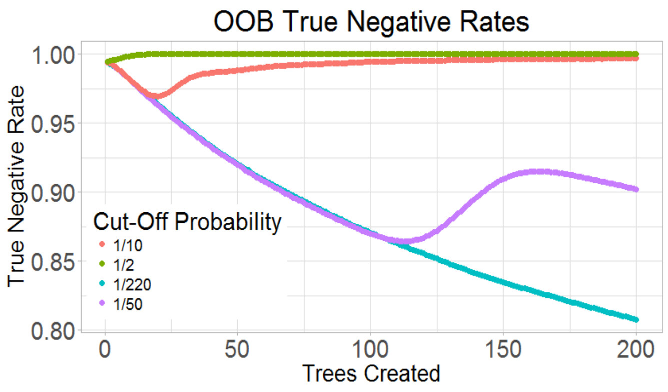


Figure 4.7: Out-of-bag true negative rate for random forests fitted to the full data set.

4.7. Including these plots here was deemed superfluous, as the results were more or less the exact same as for the random forests fitted to the full data set. The highest balanced accuracy for the random forests fitted on the full data set was obtained when $\alpha = 1/220$ and was equal to 0.778, while for the reduced model the highest BACC was 0.772 for the same value of α.

4.3.4 Logistic Regression with Resampling

Resampling methods were briefly investigated on sampled data in sections 4.2.2 and 4.2.3. In this section we apply the three resampling methods at degrees 0.05, 0.10 and 0.20 to the full data set and fit logistic models to these 9 data sets. After having been fitted, the BACC for each model was calculated for many different values of α. These are plotted in figures 4.8, 4.9 and 4.10.

In the first of these figures, we see the BACC for logistic models trained on the full data set after they have been resampled such that the imbalance in the training sets is

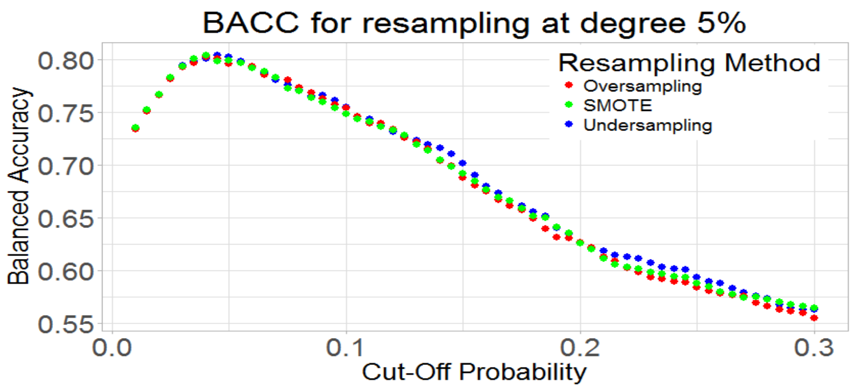


Figure 4.8: Balanced Accuracy for logistic models fitted to resampled data sets with $p_t = 0.05$.

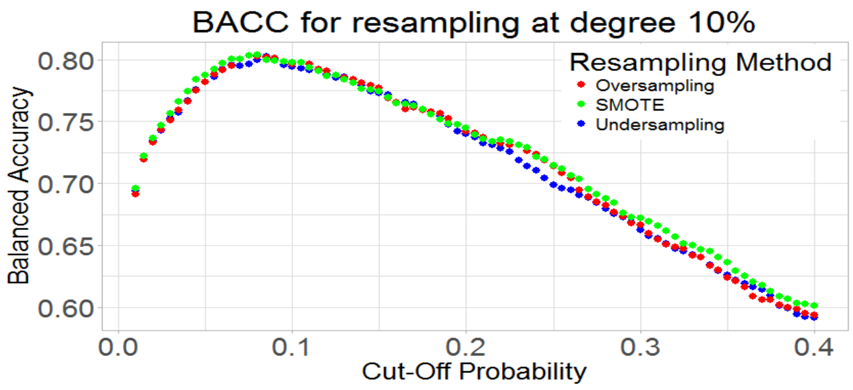


Figure 4.9: Balanced Accuracy for logistic models fitted to resampled data sets with $p_t = 0.10$.

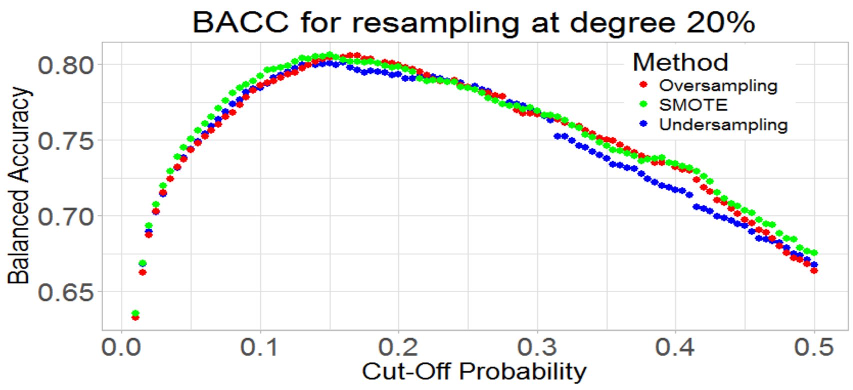


Figure 4.10: Balanced Accuracy for logistic models fitted to resampled data sets with $p_t = 0.20$.

$p_t = 0.05$. The BACC seems to peak just below $\alpha = p_t = 0.05$ for all three resampling methods. Undersampling achieved the highest BACC with a value of 0.804, with SMOTE and oversampling following closely with maximum values of 0.803 and 0.802, respectively. A few areas show the BACC of the undersampled model to lie slightly above the graphs of the two other resampling methods, roughly around $\alpha \in [0.13, 0.17]$ and $\alpha \in [0.22, 0.27]$. We note that these are minor differences, and lie well outside the area of where we can find the optimal value for α.

The second figure shows the case where the degree of resampling has been increased to 10 % for all three methods. SMOTE achieves the highest BACC for this degree of resampling with a value of 0.804. Undersampling and oversampling both achieve a maximum of 0.802, which is very similar. The peak for the BACC seems to lie slightly below 0.1, i.e below the value of p_t.

In the third figure the degree of resampling has been further increased to 20 %. SMOTE and oversampling both achieve a BACC of 0.806, while undersampling peaks at 0.801. We can see intervals for α where the BACC for the undersampled model clearly lies below the two other graphs. Oversampling and SMOTE appear to follow each other closely. We note that the original distribution had an imbalance in the data set of $p = 1/220$, while the resampling techniques here yield training sets with imbalance $p = 1/5 = 20\%$. For the undersampled training set this means that we have reduced the sample size to a fraction 1/44 of the original size. While we saw that undersampling appeared to perform just as well as SMOTE and oversampling for lower degrees of resampling, it appears that the method seems to perform slightly worse for this degree of resampling.

The issue mentioned in section 2.1.3 of the training set not containing all factor levels was encountered at this degree of resampling. An initial random seed of 4000 was used on the first attempt to create an undersampled training set of degree 20%. In **R** the command would look like this: `set.seed(4000)`. When attempting to measure the performance of this undersampled model on the test set, an error was given that stated that one of the variables (`ApplicationSalesChannel`) had unseen levels. Resampling was performed again with a random seed of 1111, and the problem was luckily circumvented.

We have found that undersampling appears to perform slightly worse on logistic models than the other resampling techniques for higher degrees of resampling. The problem of encountering new factor levels after training the model on a very small subset of the data was also encountered, and even higher degrees of resampling would have made this issue even more problematic. The optimal cut-off values for a resampled data set in all cases seemed to lie slightly below the degree of resampling p_t.

Adjusting Biased Probabilities

In section 2.1.5 we discussed a method for adjusting the probabilities produced from a classifier trained on a resampled training set. We have seen throughout this thesis that the cut-off probability α needs to be significantly less than 0.5 in order for models trained on imbalanced training sets to be able to achieve a high BACC. Applying the method presented by Latinne et al. (2001) to adjust for the bias emerging from training a model on a resampled data set would be counter-intuitive. The method would adjust the probabilities given by a resampled method down towards 0, which is the opposite of what we require from our classifier. If accuracy was our desired performance measure, then this adjustment might be useful, but we see no reason to apply it here. Accuracy is not a good measure for imbalanced data, and we therefore choose to refrain from using this method. BACC cannot be improved by applying the adjustment to the probabilities.

4.3.5 Random Forests with Resampling

The final result we would like to discuss is what happens to a random forest when we apply the algorithm to resampled data sets. In table 4.6 we see the results of fitting 9 random

Resampling Method	Degree	α	TN	FP	FN	TP	BACC
Undersampling	5 %	0.05	113157	36161	91	591	0.812
Undersampling	10%	0.10	112436	36882	81	601	0.817
Undersampling	20%	0.20	113418	35900	83	599	0.819
Oversampling	5 %	0.05	142180	7138	501	181	0.609
Oversampling	10%	0.10	147281	2037	633	49	0.529
Oversampling	20%	0.20	148975	343	676	6	0.503
SMOTE	5 %	0.05	137797	11521	434	248	0.643
SMOTE	10%	0.10	141692	7626	506	176	0.603
SMOTE	20%	0.20	144625	4693	570	112	0.566

Table 4.6: Performance of random forests. The classifiers are trained on the reduced data set, and resampled by all three resampling methods, at degrees 0.05, 0.1 and 0.2. Cut-off equal to the degree of resampling.

forests to the reduced data set, with the same resampling as in the previous section. Two hundred decision trees were used in each forest. Random forests cannot be applied to test data with varying cut-off probability after they have been trained, as we have done for logistic regression when producing figures 4.8, 4.9 and 4.10. Finding the optimal value for α will be an extremely time-consuming process, which is why it was decided to set $\alpha = p_t$ in all cases. While all the resampling techniques seemed to yield quite similar results in all cases when applied to logistic regression, we here find that the results are not at all similar. From the table, we see that for the given values of α undersampling is vastly superior to both oversampling and SMOTE when it comes to optimizing the BACC. The true positive rate for undersampling is much higher than for SMOTE and oversampling. Undersampling has the added bonus of being trained much faster than the other methods, due to the fact the training set will be much smaller than in the other cases. This is further increased by the fact that random forests take longer to converge when α is close to 0, and the values for α have to be adjusted down in order for SMOTEd and oversampled random forests to be able to correctly identify the minority class. However, it is important to point out that this does not directly imply that undersampling is a better method when applied to random forests than the other resampling methods. It merely implies that if the BACC is to be optimized, the cut-off probability must be set lower for oversampling and SMOTE than for undersampling.

Chapter 5

Concluding Remarks

Here, the findings from working on this thesis is summarized, and recommendations for further work are provided.

5.1 Summary of the Results

The results were split into two main sections: sampled data and data provided by Sparebank 1. The code in the appendix make the results from the sampled data entirely reproducible.

We have found that undersampling, oversampling and SMOTE all have the ability to increase the performance of a classifier. Undersampling is prone to erratic behavior when the number of minority observations is small, while this has not proven to be a problem for oversampling and SMOTE. Undersampling seems to perform slightly worse when the degree of resampling increases, which is assumed to be because the training set gets reduced in size when this is done. Undersampling is superior to the other methods when issues of memory constraints and time constraints are a factor.

We found that the optimal value for the cut-off probability α for a given classifier seemed to be equal to the imbalance of the data set the classifier was trained on, when performance was measured by balanced accuracy. Interestingly, random forests did not exhibit this quality: when using oversampling and SMOTE on an imbalanced data set and applying a random forest to the resampled data, it was found that α had to be adjusted down significantly in order for the random forest to achieve the same balanced accuracy as when undersampling was the chosen resampling method. Random forests were found to take a long time to converge when α was small, which is necessary when attempting to classify an imbalanced data set.

Adjustment of the probabilities produced from classifiers trained on resampled data sets by the method presented in Latinne et al. (2001) was considered, and discarded. The method was determined to not be suitable when BACC was the performance measure to be maximized.

5.2 Recommendations for Further Work

There are probably many things that could have been done differently in this thesis, for better or for worse. Balanced Accuracy was chosen as the desired performance measure of the classifiers. There are many arguments to be made on what constitutes a good classifier, and balanced accuracy might be suitable in some situations, and useless in others. Other performance measures might yield very different results to what has been found here. There are many classification problems that are even more imbalanced than the one considered in this thesis, for instance in medicine. These may have life and death consequences, and in this case one could look at the generalized version of the BACC, which is given in the theory section. This performance measure was never investigated, but here it would make sense to heavily emphasize the true positive rate, in order to not miss a patient who has an undiscovered disease.

When sampling data, it was attempted to use a categorical variable and see how well it could help predict the response as the imbalance in the data set became more pronounced. Code for this was written and is given in the appendix under `sampling.R`, but there were a few issues in providing meaningful results that made this attempt fail. Our sampled data only came from a mixed normal distribution, so in a future study similar to this thesis one could look at different distributions to see how the results change.

For the data set provided by Sparebank 1, one could try different classification algorithms to see if there are any that are capable of producing better results than the ones found here. Examples could be boosting algorithms and support vector machines. Further analysis could go into what kinds of variables are good for determining the response in such a data set.

Bibliography

Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U., 1999. When is “nearest neighbor” meaningful? In: Database Theory – ICDT’99. Springer Berlin Heidelberg, pp. 217–235.

Breiman, L., 2001. Random forests. Machine Learning 45 (1), 5–32.

Breiman, L., 2002. Manual On Setting Up, Using, And Understanding Random Forests V3.1.

Brier, G. W., 1950. Verification of forecasts expressed in terms of probability. Monthly Weather Review 78 (1), 1–3.

Casella, G., Berger, R., 2002. Statistical Inference. Brooks/Cole, Cengage Learning.

Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P., 2002. Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research 16 (1), 321–357.

Dilworth, K., 2019. Rate survey: Average card apr remains at 17.71 percent. Accessed 22. August, 2019.
URL `https://bit.ly/2Hpxf9T`

Everitt, B. S., Hand, D. J., 1981. Finite Mixture Distributions, 1st Edition. Chapman and Hall.

Fahrmeir, L., Kneib, T., Lang, S., Marx, B., 2013. Regression Models, Methods and Applications. Springer.

Finansdepartementet, February 2019. Forskrift om krav til finansforetakenes utlånspraksis for forbrukslån.
URL `https://bit.ly/2GBakLI`

Frogner, T. B., 2019. Predicting refinancing of credit card debt. Unpublished project thesis at NTNU.

Latinne, P., Saerens, M., Decaestecker, C., 01 2001. Adjusting the outputs of a classifier to new a priori probabilities may significantly improve classification accuracy: Evidence from a multi-class problem in remote sensing. In: ICML. pp. 298–305.

Lujan-Moreno, G. A., Howard, P., Rojas, O., Montgomery, D., 2018. Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study. Expert Systems With Applications 109, 195–205.

Muchai, E., Odongo, L., 2014. Comparison of crisp and fuzzy classification trees using gini index impurity measure on simulated data. European Scientific Journal 10 (18), 130–134.

Rahman, M. M., Davis, D., 2013. Addressing the class imbalance problem in medical datasets. International Journal of Machine Learning and Computing 3 (2), 224–228.

Chapter 6

Appendix

Appendix A

Variables in the data set provided by Sparebank 1, with descriptions.

Variable	Description
BK_ACCOUNT_ID	Account number, anonymized
PeriodId	Date on format YYYYMMDD
Date	Date on format YYYY-MM-DD
YearMonth	Year and month on format YYYYMM
PNRSerial	Digits 7 and 8 in national identification number
CustomerAge	Customer's age in years
MonthsSinceAccountCreated	Account's age in months
PRODUCT_NAME	Name of product (card type)
STATEMENT_DUE_DAY_OF_MONTH_NUM	Chosen due date (5.,10.,15. or 20.)
ApplicationSalesChannel	Channel of application and / or sale
CAMPAIGN_NAME	Campaign (if any)
CLOSING_BALANCE_AMT	Total amount printed on last statement
DISTRIBUTOR_NAME	Name of distributing bank
GENDER_NAME	Gender
HAS_DIRECT_DEBIT_AGREEMENT_IND	Indicator, direct debit agreement selected
HAS_ESTATEMENT_AGREEMENT_IND	Indicator, e-statement selected
average_credit_limit_last12	Average credit limit last 12 months
average_revolvingbalance_last12	Average revolving balance last 12 months
avg_rev_bal_L3M	Average revolving balance last 3 months
rev_uti_currmth	Revolving balance divided by credit limit this month

avg_payment_L3M	Average payment last 3 months
rev_per_uti_change_L3M	Change in revolving utilization (revolving balance divided by credit limit) last 3 months
MonthEnd_uti_Change	Change in revolving utilization by end of month
payment_amt_change_L3M	Change in payment amount last 3 months
RevUti12	Average revolving balance last 12 months divided by average credit limit last 12 months
AvgRevBalL3onL12	(Average revolving balance last 3 months divided by average credit limit last 3 months) divided by (Average revolving balance last 12 months divided by average credit limit last 12 months)
QCashpartL12	Part of sum of transactions in class Quasi Cash last 12 months
QCashpartL3	Part of sum of transactions in class Quasi Cash last 3 months
QCashL3onL12	(Part of sum of transactions in class Quasi Cash last 3 months) divided by (Part of sum of transactions in class Quasi Cash last 12 months)
TravelpartL12	Sum of transactions in classes, Airline, Hotel motel and other transport last 12 months divided by sum of transactions in all classes last 12 months
TravelpartL3	Sum of transactions in classes, Airline, Hotel motel and other transport last 12 months divided by sum of transactions in all classes last 3 months
Segment9Name	Segment name with 9 segments (originally 9 segments, but a few more have been added recently making it 11 segments)
Segment23Name	Segment name with 23 segments originally, now 25
Score	Simple risk score between 0 and 7
SUM_of_CreditLimitIncreaseFlag	Number of credit limit increases last 12 months
SUM_of_CreditLimitDecreaseFlag	Number of credit limit decreases last 12 months
SUM_of_PaymentOverDueFlag	Number of months with payment overdue last 12 months
SUM_of_FirstDunningFlag	Number of months with dunning last 12 months
SUM_of_CollectionAdviceFlag	Number of months with collection advice last 12 months
SUM_of_CollectionFlag	Number of months with debt collection last 12 months
SUM_of_CardFraudFlag	Number of months with card fraud flag (transactions marked as possible fraud) last 12 months
SUM_of_CardLostFlag	Number of months with card lost flag (card marked as lost) last 12 months
SUM_of_CardStolenFlag	Number of months with card stolen flag (card marked as stolen) last 12 months

SUM_of_AIRLINEL12	Sum of transactions in given class last 12 months
SUM_of_CLOTHING_STORESL12	Sum of transactions in given class last 12 months
SUM_of_FOOD_STORES_WAREHOUSEL12	Sum of transactions in given class last 12 months
SUM_of_HOTEL_MOTELL12	Sum of transactions in given class last 12 months
SUM_of_HARDWAREL12	Sum of transactions in given class last 12 months
SUM_of_INTERIOR_FURNISHINGSL12	Sum of transactions in given class last 12 months
SUM_of_OTHER_RETAILL12	Sum of transactions in given class last 12 months
SUM_of_OTHER_SERVICESL12	Sum of transactions in given class last 12 months
SUM_of_OTHER_TRANSPORTL12	Sum of transactions in given class last 12 months
SUM_of_RECREATIONL12	Sum of transactions in given class last 12 months
SUM_of_RESTAURANTS_BARSL12	Sum of transactions in given class last 12 months
SUM_of_SPORTING_TOY_STORESL12	Sum of transactions in given class last 12 months
SUM_of_TRAVEL_AGENCIESL12	Sum of transactions in given class last 12 months
SUM_of_VEHICLESL12	Sum of transactions in given class last 12 months
SUM_of_QUASI_CASHL12	Sum of transactions in given class last 12 months
SUM_of_AIRLINEL3	Sum of transactions in given class last 3 months
SUM_of_CLOTHING_STORESL3	Sum of transactions in given class last 3 months
SUM_of_FOOD_STORES_WAREHOUSEL3	Sum of transactions in given class last 3 months
SUM_of_HOTEL_MOTELL3	Sum of transactions in given class last 3 months
SUM_of_HARDWAREL3	Sum of transactions in given class last 3 months
SUM_of_INTERIOR_FURNISHINGSL3	Sum of transactions in given class last 3 months
SUM_of_OTHER_RETAILL3	Sum of transactions in given class last 3 months
SUM_of_OTHER_SERVICESL3	Sum of transactions in given class last 3 months
SUM_of_OTHER_TRANSPORTL3	Sum of transactions in given class last 3 months
SUM_of_RECREATIONL3	Sum of transactions in given class last 3 months
SUM_of_RESTAURANTS_BARSL3	Sum of transactions in given class last 3 months
SUM_of_SPORTING_TOY_STOR0ESL3	Sum of transactions in given class last 3 months
SUM_of_TRAVEL_AGENCIESL3	Sum of transactions in given class last 3 months
SUM_of_VEHICLESL3	Sum of transactions in given class last 3 months
SUM_of_QUASI_CASHL3	Sum of transactions in given class last 3 months
SHORT_RULE_DESC	Short description of the account's finances
lead1YearMonth	YearMonth+1
lead2YearMonth	YearMonth+2
lead3YearMonth	YearMonth+3
ReFinInd	The response, which equals "Does the customer receive refinancing from a competitor during the next 3 months after the given month?" 0 = no, 1 = yes.

Appendix B

Print out can be found in this appendix.

6.1 Logistic Model on all variables

This is a summary of the a model fitted on (almost) all variables, before the data preparation was complete. A subset of the data was used, and the coefficients will not be entirely equal to what was shown in the dataprep section.

```
> summary(modelUseless)

Call:
glm(formula = ReFinInd ~ ., family = "binomial", data = dataSmall)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-8.4904  -0.0526   0.0000   0.0000   4.2723

Coefficients: (1 not defined because of singularities)
                                                          Estimate Std. Error  z value Pr(>|z|)
(Intercept)                                             -5.185e+01  3.333e+04   -0.002  0.99876
PNRSerial1                                               1.923e+01  3.235e+04    0.001  0.99953
PNRSerial2                                               1.947e+01  3.235e+04    0.001  0.99952
PNRSerial3                                               1.942e+01  3.235e+04    0.001  0.99952
PNRSerial4                                               1.952e+01  3.235e+04    0.001  0.99952
CustomerAge                                             -1.764e-02  7.859e-03   -2.245  0.02480 *
MonthsSinceAccountCreated                               -3.855e-03  3.575e-03   -1.078  0.28099
PRODUCT_NAMELOfavør MasterCard                           4.801e-01  1.021e+00    0.470  0.63813
PRODUCT_NAMESB1 EXTRA MC                                -7.740e-01  1.245e+00   -0.622  0.53417
PRODUCT_NAMESB1 GOLD MC                                 -1.141e+00  7.126e-01   -1.602  0.10918
PRODUCT_NAMESB1 UNG MC                                  -2.142e+01  6.584e+04    0.000  0.99974
PRODUCT_NAMESH GOLD MC                                  -1.117e+00  7.233e-01   -1.544  0.12258
PRODUCT_NAMESparebank 1 Platinum MC                     -2.176e+01  1.118e+04   -0.002  0.99845
PRODUCT_NAMESpareBank 1 Visa Business Card               5.391e+00  3.562e+05    0.000  0.99999
PRODUCT_NAMESpareBank 1 Visa Gold                       -1.982e+01  3.208e+04   -0.001  0.99951
STATEMENT_DUE_DAY_OF_MONTH_NUM10                        -4.314e-01  3.804e-01   -1.134  0.25675
STATEMENT_DUE_DAY_OF_MONTH_NUM15                        -6.152e-02  2.495e-01   -0.247  0.80523
STATEMENT_DUE_DAY_OF_MONTH_NUM20                         6.908e-02  2.409e-01    0.287  0.77426
ApplicationSalesChannelAutentisert web                   5.998e-02  6.766e-01    0.089  0.92936
ApplicationSalesChannelNettbank                          2.387e-01  3.661e-01    0.652  0.51436
ApplicationSalesChannelOpen web                          2.151e+03  3.001e+07    0.000  0.99994
ApplicationSalesChannelOperatørkanal                    -1.581e-01  3.651e-01   -0.433  0.66501
ApplicationSalesChannelResponsside                       7.933e+11  1.704e+13    0.047  0.96288
CAMPAIGN_NAME201602-Nysalgskampanje MasterCard Extra kort       -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAME201603-Nysalgskampanje alle banker         -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAME201603-Nysalgskampanje alle banker påminnelse      -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAME201603-Nysalgskampanje BN Bank             -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAME201603-Nysalgskampanje BN Bank Påminnelse  -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAME201604-Nysalgkampanje post alle banker     -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe A - A       -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe A - B       -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe C - A       -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe C - B       -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe C Chall - A -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe C Chall - B -4.504e+15  1.704e+13 -264.297  < 2e-16 ***
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe D Post - A  -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15- Nysalgskampanje SR-Bank- Gruppe D Post - B  -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje BN-Bank B            -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje BN Bank A            -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje BN Bank A Reminder   -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje BN Bank B Reminder   -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje Hedmark - A          -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje Hedmark - B          -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje Hedmark -Challenger A        -7.933e+11  1.704e+13   -0.047  0.96288
CAMPAIGN_NAMENov15-Nysalgskampanje Hedmark -Challenger B        -7.933e+11  1.704e+13   -0.047  0.96288
```

```
CAMPAIGN_NAMENysalg SR-Bank (løpende)                          -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgs kampanje alle banker                      -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje - Hedmark                         -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje alle banker                       -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje alle banker - A                   -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje alle banker - B                   -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje alle banker - Påminnelse          -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje alle banker - påminnelse post     -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje BN Bank                           -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje BN Bank - Påminnelse              -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje Post Hedmark                      -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje SMN                               -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMENysalgskampanje SR-bank                           -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMEOkt15- Nysalgskampanje Alle påminnelse            -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMEOkt15- Nysalgskampanje Alle påminnelse post       -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMEOkt15- Nysalgskampanje SMN påminnelse             -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMEOkt15- Nysalgskampanje SMN påminnelse post        -7.933e+11  1.704e+13  -0.047  0.96288
CAMPAIGN_NAMEOkt15-Reminder SR-Bank post                       -7.933e+11  1.704e+13  -0.047  0.96288
CLOSING_BALANCE_AMT                                            -3.958e-05  1.279e-05  -3.095  0.00197 **
DISTRIBUTOR_NAMESpareBank 1 BV                                  2.385e-01  4.674e-01   0.510  0.60985
DISTRIBUTOR_NAMESpareBank 1 Gudbrandsdal                       -2.065e+01  1.440e+04  -0.001  0.99886
DISTRIBUTOR_NAMESpareBank 1 Hallingdal Valdres                 -2.032e+01  1.252e+04  -0.002  0.99870
DISTRIBUTOR_NAMESpareBank 1 Kredittkort                         1.222e+00  8.133e+04   0.000  0.99999
DISTRIBUTOR_NAMESpareBank 1 Lom og Skjåk                        3.272e-01  1.066e+00   0.307  0.75887
DISTRIBUTOR_NAMESpareBank 1 Modum                              -2.043e+01  1.159e+04  -0.002  0.99859
DISTRIBUTOR_NAMESpareBank 1 Nord-Norge                         -3.893e-01  3.846e-01  -1.012  0.31140
DISTRIBUTOR_NAMESpareBank 1 Nordvest                            5.808e-01  5.490e-01   1.058  0.29008
DISTRIBUTOR_NAMESpareBank 1 Ringerike Hadeland                  1.929e-01  5.923e-01   0.326  0.74468
DISTRIBUTOR_NAMESpareBank 1 SMN                                 5.031e-01  3.400e-01   1.480  0.13896
DISTRIBUTOR_NAMESpareBank 1 SR-Bank                             1.029e-01  3.591e-01   0.287  0.77445
DISTRIBUTOR_NAMESpareBank 1 Søre Sunnmøre                      -3.206e-01  1.060e+00  -0.303  0.76222
DISTRIBUTOR_NAMESpareBank 1 Telemark                            2.374e-01  4.943e-01   0.480  0.63107
DISTRIBUTOR_NAMESpareBank 1 Østfold Akershus                    5.980e-01  4.341e-01   1.378  0.16835
DISTRIBUTOR_NAMESpareBank 1 Østlandet                                  NA         NA      NA       NA
GENDER_NAMEMann                                                 1.289e-01  1.838e-01   0.701  0.48301
HAS_DIRECT_DEBIT_AGREEMENT_IND1                                 1.969e-01  1.920e-01   1.026  0.30502
HAS_ESTATEMENT_AGREEMENT_IND1                                   1.187e-02  1.825e-01   0.065  0.94812
average_credit_limit_last12                                     1.003e-05  7.822e-06   1.283  0.19959
average_revolvingbalance_last12                                -2.341e-05  1.313e-05  -1.782  0.07469 .
avg_rev_bal_L3M                                                -1.491e-05  1.571e-05  -0.949  0.34278
rev_uti_currmth                                                 8.544e-03  4.493e-03   1.902  0.05722 .
avg_payment_L3M                                                 3.997e-07  1.218e-04   0.003  0.99738
rev_per_uti_change_L3M                                          3.090e-04  6.715e-04   0.460  0.64538
MonthEnd_uti_Change                                            -1.405e-07  2.052e-07  -0.685  0.49355
payment_amt_change_L3M                                          7.186e-07  7.171e-06   0.100  0.92017
RevUti12                                                        9.547e-01  7.785e-01   1.226  0.22007
AvgRevBalL3onL12                                                3.514e-02  6.599e-02   0.532  0.59439
QCashpartL12                                                   -2.079e-01  4.252e-01  -0.489  0.62495
QCashpartL3                                                    -1.973e-01  4.624e-01  -0.427  0.66954
QCashL3onL12                                                    1.189e-02  8.029e-03   1.481  0.13861
TravelpartL12                                                  -5.159e-01  6.447e-01  -0.800  0.42363
TravelpartL3                                                    7.355e-01  5.426e-01   1.355  0.17526
Segment9NameDelinquent                                          2.716e+01  8.027e+03   0.003  0.99730
Segment9NameLast active 4-6 mths ago                            9.645e+00  1.017e+04   0.001  0.99924
Segment9NameLast active 7-12 mths ago                          -1.840e+01  1.013e+04  -0.002  0.99855
Segment9NameNot active in last 12 mths                          7.895e-01  9.692e+03   0.000  0.99994
Segment9NameOccasional Revolver                                 2.756e+01  8.027e+03   0.003  0.99726
Segment9NameRevolved only                                       2.743e+01  8.027e+03   0.003  0.99727
Segment9NameRevolver                                            2.779e+01  8.027e+03   0.003  0.99724
Segment9NameTransactor                                          2.663e+01  8.027e+03   0.003  0.99735
Score                                                           1.025e-01  9.574e-02   1.071  0.28421
SUM_of_CreditLimitIncreaseFlag                                  1.608e-01  2.256e-01   0.713  0.47590
SUM_of_CreditLimitDecreaseFlag                                  1.492e-01  4.439e-01   0.336  0.73673
SUM_of_PaymentOverDueFlag                                       1.106e-01  2.375e-01   0.466  0.64134
SUM_of_FirstDunningFlag                                        -3.030e-01  2.404e-01  -1.260  0.20763
SUM_of_CollectionAdviceFlag                                     2.160e-01  2.928e-01   0.738  0.46066
SUM_of_CollectionFlag                                          -4.302e-01  4.064e-01  -1.059  0.28981
SUM_of_CardFraudFlag                                           -8.010e-01  1.038e+00  -0.772  0.44019
SUM_of_CardLostFlag                                             9.224e-04  4.750e-01   0.002  0.99845
SUM_of_CardStolenFlag                                           1.096e+00  9.142e-01   1.198  0.23076
SUM_of_AIRLINEL12                                              -1.054e-04  7.030e-05  -1.499  0.13395
SUM_of_CLOTHING_STORESL12                                       3.267e-05  2.609e-05   1.252  0.21060
SUM_of_FOOD_STORES_WAREHOUSEL12                                 2.140e-05  2.123e-05   1.008  0.31355
SUM_of_HOTEL_MOTELL12                                           7.213e-06  3.491e-05   0.207  0.83630
SUM_of_HARDWAREL12                                             -1.250e-05  4.170e-05  -0.300  0.76442
SUM_of_INTERIOR_FURNISHINGSL12                                  2.979e-05  1.477e-05   2.017  0.04369 *
SUM_of_OTHER_RETAILL12                                         -6.563e-05  4.893e-05  -1.341  0.17979
SUM_of_OTHER_SERVICESL12                                       -4.185e-05  5.512e-05  -0.759  0.44769
SUM_of_OTHER_TRANSPORTL12                                       2.522e-05  6.389e-05   0.395  0.69304
SUM_of_RECREATIONL12                                            2.964e-05  4.495e-05   0.659  0.50961
SUM_of_RESTAURANTS_BARSL12                                     -1.085e-04  6.496e-05  -1.670  0.09500 .
SUM_of_SPORTING_TOY_STORESL12                                  -6.157e-06  6.336e-05  -0.097  0.92259
SUM_of_TRAVEL_AGENCIESL12                                      -8.817e-05  5.808e-05  -1.518  0.12897
SUM_of_VEHICLESL12                                             -7.564e-05  5.011e-05  -1.509  0.13120
SUM_of_QUASI_CASHL12                                            1.031e-05  8.363e-06   1.233  0.21775
SUM_of_AIRLINEL3                                               -6.937e-05  1.812e-04  -0.383  0.70184
SUM_of_CLOTHING_STORESL3                                       -3.046e-04  1.586e-04  -1.921  0.05475 .
SUM_of_FOOD_STORES_WAREHOUSEL3                                 -5.581e-06  7.340e-05  -0.076  0.93940
SUM_of_HOTEL_MOTELL3                                           -2.385e-04  1.689e-04  -1.412  0.15785
SUM_of_HARDWAREL3                                              -1.686e-05  9.478e-05  -0.178  0.85879
```

```
SUM_of_INTERIOR_FURNISHINGSL3                                      2.012e-05  3.238e-05    0.621  0.53435
SUM_of_OTHER_RETAILL3                                              1.531e-05  1.141e-04    0.134  0.89328
SUM_of_OTHER_SERVICESL3                                            8.346e-05  8.452e-05    0.987  0.32342
SUM_of_OTHER_TRANSPORTL3                                          -1.713e-05  1.387e-04   -0.124  0.90170
SUM_of_RECREATIONL3                                               -1.132e-04  2.129e-04   -0.532  0.59499
SUM_of_RESTAURANTS_BARSL3                                          2.333e-04  9.821e-05    2.375  0.01753 *
SUM_of_SPORTING_TOY_STOR0ESL3                                     -1.714e-04  2.461e-04   -0.697  0.48602
SUM_of_TRAVEL_AGENCIESL3                                           1.599e-05  1.268e-04    0.126  0.89970
SUM_of_VEHICLESL3                                                  1.022e-04  7.593e-05    1.346  0.17845
SUM_of_QUASI_CASHL3                                                7.232e-06  2.146e-05    0.337  0.73611
SHORT_RULE_DESCDCA 24 To 36 Ind                                   -2.116e+01  2.064e+04   -0.001  0.99918
SHORT_RULE_DESCExeeded Num Of Dunnings 12 Ind                      6.294e-01  9.942e-01    0.633  0.52667
SHORT_RULE_DESCHad Collection Adv 6 To 12 Ind                      5.701e-01  8.268e-01    0.690  0.49050
SHORT_RULE_DESCHad Neg Status 0 to 6 Ind                          -5.513e-02  7.579e-01   -0.073  0.94201
SHORT_RULE_DESCHas Balance Above CL Ind                            5.299e-01  6.616e-01    0.801  0.42317
SHORT_RULE_DESCHas Missed Payments Ind                             4.394e-01  7.803e-01    0.563  0.57337
SHORT_RULE_DESCHas Neg Status Ind                                  4.182e-01  1.277e+00    0.327  0.74337
SHORT_RULE_DESCHas Too High Total Limit Ind                       -9.671e-01  9.006e-01   -1.074  0.28290
SHORT_RULE_DESCIs Above Auth Limit Ind                            -9.085e-01  1.210e+00   -0.751  0.45286
SHORT_RULE_DESCIs At DCA Ind                                       8.225e-01  1.982e+00    0.415  0.67817
SHORT_RULE_DESCIs On Black List Ind                                1.111e+00  9.764e-01    1.138  0.25523
SHORT_RULE_DESCIs White Credit Balance Ind                         2.693e-02  1.202e+00    0.022  0.98213
SHORT_RULE_DESCIs White Debet Balance Ind                          4.578e-01  6.300e-01    0.727  0.46736
SHORT_RULE_DESCIs White No Balance Ind                            -1.888e+01  4.232e+03   -0.004  0.99644
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1964.1  on 39999  degrees of freedom
Residual deviance: 1743.7  on 39842  degrees of freedom
AIC: 2059.7

Number of Fisher Scoring iterations: 25
```

6.2 The Full Logistic Model

Summary of the full logistic model, trained on the original data with no resampling.

```
> summary(model1)

Call:
glm(formula = ReFinInd ~ ., family = "binomial", data = dataset[training,
    ])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.3332  -0.0963  -0.0402  -0.0181   4.4169

Coefficients:
                                           Estimate Std. Error z value Pr(>|z|)
(Intercept)                              -7.734e+00  1.233e+00  -6.275 3.50e-10 ***
CustomerAge                              -1.487e-02  2.392e-03  -6.216 5.10e-10 ***
MonthsSinceAccountCreated                -2.550e-03  8.236e-04  -3.096  0.00196 **
STATEMENT_DUE_DAY_OF_MONTH_NUM10         -2.091e-01  1.049e-01  -1.993  0.04626 *
STATEMENT_DUE_DAY_OF_MONTH_NUM15         -7.384e-02  7.513e-02  -0.983  0.32565
STATEMENT_DUE_DAY_OF_MONTH_NUM20         -7.075e-02  7.657e-02  -0.924  0.35548
ApplicationSalesChannelAutentisert web    2.611e-01  1.460e-01   1.788  0.07385 .
ApplicationSalesChannelNettbank           1.388e-02  9.639e-02   0.144  0.88546
ApplicationSalesChannelOpen web          -8.887e+00  1.251e+02  -0.071  0.94338
ApplicationSalesChannelOperatørkanal     -9.083e-02  9.213e-02  -0.986  0.32418
ApplicationSalesChannelResponsside        2.367e-01  1.186e-01   1.996  0.04596 *
CLOSING_BALANCE_AMT                      -2.416e-05  3.029e-06  -7.977 1.50e-15 ***
GENDER_NAMEMann                           1.156e-01  5.718e-02   2.022  0.04314 *
HAS_DIRECT_DEBIT_AGREEMENT_IND1           9.024e-02  6.331e-02   1.425  0.15406
HAS_ESTATEMENT_AGREEMENT_IND1             1.416e-01  5.720e-02   2.476  0.01328 *
average_credit_limit_last12               4.088e-06  8.517e-07   4.799 1.59e-06 ***
average_revolvingbalance_last12          -1.920e-05  3.528e-06  -5.440 5.32e-08 ***
avg_rev_bal_L3M                           9.680e-07  3.295e-06   0.294  0.76893
rev_uti_currmth                           2.502e-03  7.959e-04   3.143  0.00167 **
avg_payment_L3M                           8.310e-06  7.239e-06   1.148  0.25099
rev_per_uti_change_L3M                    1.047e-06  3.386e-06   0.309  0.75707
MonthEnd_uti_Change                      -5.192e-11  4.342e-08  -0.001  0.99905
payment_amt_change_L3M                   -3.337e-09  4.763e-08  -0.070  0.94415
RevUti12                                  1.034e+00  1.960e-01   5.274 1.33e-07 ***
AvgRevBalL3onL12                         -5.644e-03  1.924e-02  -0.293  0.76922
QCashpartL12                              4.745e-02  1.315e-01   0.361  0.71814
QCashpartL3                              -5.163e-02  1.401e-01  -0.368  0.71250
QCashL3onL12                              1.936e-04  2.025e-03   0.096  0.92382
TravelpartL12                            -4.204e-01  1.910e-01  -2.201  0.02774 *
TravelpartL3                             -3.899e-02  1.741e-01  -0.224  0.82280
```

```
Segment9NameDelinquent                        1.440e+00  1.140e+00   1.263  0.20671
Segment9NameEMOB - Active in last 6 mths      1.793e+00  1.217e+00   1.474  0.14052
Segment9NameEMOB - Not active last 6 mths     3.563e-01  1.582e+00   0.225  0.82180
Segment9NameLast active 4-6 mths ago         -8.289e-01  1.574e+00  -0.527  0.59837
Segment9NameLast active 7-12 mths ago        -6.516e-01  1.573e+00  -0.414  0.67860
Segment9NameOccasional Revolver               2.156e+00  1.204e+00   1.790  0.07351 .
Segment9NameRevolved only                     1.232e+00  1.223e+00   1.007  0.31391
Segment9NameRevolver                          1.872e+00  1.205e+00   1.554  0.12015
Segment9NameTransactor                        3.559e-01  1.220e+00   0.292  0.77040
Score                                         2.162e-01  2.859e-02   7.562 3.96e-14 ***
SUM_of_CreditLimitIncreaseFlag                3.688e-01  6.305e-02   5.849 4.94e-09 ***
SUM_of_CreditLimitDecreaseFlag               -1.573e-01  1.370e-01  -1.148  0.25084
SUM_of_PaymentOverDueFlag                    -6.784e-02  7.111e-02  -0.954  0.34005
SUM_of_FirstDunningFlag                       1.583e-02  7.094e-02   0.223  0.82341
SUM_of_CollectionAdviceFlag                   2.641e-02  9.079e-02   0.291  0.77118
SUM_of_CollectionFlag                        -4.645e-02  1.138e-01  -0.408  0.68310
SUM_of_CardFraudFlag                          1.542e-01  1.945e-01   0.793  0.42797
SUM_of_CardLostFlag                           1.100e-01  1.238e-01   0.888  0.37441
SUM_of_CardStolenFlag                         2.331e-01  3.803e-01   0.613  0.53994
SUM_of_AIRLINEL12                            -1.283e-05  8.758e-06  -1.465  0.14299
SUM_of_CLOTHING_STORESL12                    -1.283e-05  8.912e-06  -1.440  0.15000
SUM_of_FOOD_STORES_WAREHOUSEL12              -3.018e-06  5.766e-06  -0.523  0.60065
SUM_of_HOTEL_MOTELL12                        -1.078e-05  8.717e-06  -1.237  0.21611
SUM_of_HARDWAREL12                            3.105e-06  7.281e-06   0.426  0.66981
SUM_of_INTERIOR_FURNISHINGSL12                7.408e-06  5.444e-06   1.361  0.17361
SUM_of_OTHER_RETAILL12                       -3.454e-06  7.130e-06  -0.484  0.62804
SUM_of_OTHER_SERVICESL12                     -4.492e-07  9.584e-06  -0.047  0.96262
SUM_of_OTHER_TRANSPORTL12                    -1.302e-05  1.570e-05  -0.829  0.40716
SUM_of_RECREATIONL12                          2.044e-06  1.513e-05   0.135  0.89252
SUM_of_RESTAURANTS_BARSL12                    1.164e-06  9.222e-06   0.126  0.89957
SUM_of_SPORTING_TOY_STORESL12                -1.611e-05  1.534e-05  -1.050  0.29364
SUM_of_TRAVEL_AGENCIESL12                    -4.605e-06  6.307e-06  -0.730  0.46527
SUM_of_VEHICLESL12                           -1.956e-06  7.398e-06  -0.264  0.79151
SUM_of_QUASI_CASHL12                         -1.859e-06  2.946e-06  -0.631  0.52799
SUM_of_AIRLINEL3                             -6.181e-06  2.252e-05  -0.274  0.78374
SUM_of_CLOTHING_STORESL3                      8.869e-06  2.284e-05   0.388  0.69776
SUM_of_FOOD_STORES_WAREHOUSEL3                2.676e-05  1.552e-05   1.724  0.08472 .
SUM_of_HOTEL_MOTELL3                          2.338e-05  1.905e-05   1.228  0.21958
SUM_of_HARDWAREL3                            -7.588e-07  1.584e-05  -0.048  0.96180
SUM_of_INTERIOR_FURNISHINGSL3                -1.074e-05  1.371e-05  -0.784  0.43323
SUM_of_OTHER_RETAILL3                         4.679e-06  1.901e-05   0.246  0.80562
SUM_of_OTHER_SERVICESL3                      -1.642e-05  2.782e-05  -0.590  0.55507
SUM_of_OTHER_TRANSPORTL3                      5.526e-05  2.421e-05   2.282  0.02247 *
SUM_of_RECREATIONL3                          -2.238e-05  4.189e-05  -0.534  0.59315
SUM_of_RESTAURANTS_BARSL3                    -2.059e-05  2.710e-05  -0.760  0.44740
SUM_of_SPORTING_TOY_STOR0ESL3                 9.471e-06  3.391e-05   0.279  0.78002
SUM_of_TRAVEL_AGENCIESL3                     -5.068e-06  1.671e-05  -0.303  0.76162
SUM_of_VEHICLESL3                             1.744e-05  1.570e-05   1.111  0.26665
SUM_of_QUASI_CASHL3                           1.705e-05  8.231e-06   2.071  0.03837 *
SHORT_RULE_DESCDCA 24 To 36 Ind              -5.619e-01  4.009e-01  -1.402  0.16104
SHORT_RULE_DESCExeeded Num Of Dunnings 12 Ind -3.927e-01  3.442e-01  -1.141  0.25390
SHORT_RULE_DESCHad Collection Adv 6 To 12 Ind -2.736e-01  2.748e-01  -0.996  0.31940
SHORT_RULE_DESCHad Neg Status 0 to 6 Ind     -8.679e-02  2.139e-01  -0.406  0.68487
SHORT_RULE_DESCHas Balance Above CL Ind       2.261e-01  1.984e-01   1.140  0.25438
SHORT_RULE_DESCHas Missed Payments Ind       -2.492e-01  2.394e-01  -1.041  0.29798
SHORT_RULE_DESCHas Neg Status Ind            -1.067e-01  4.059e-01  -0.263  0.79272
SHORT_RULE_DESCHas Too High Total Limit Ind  -6.618e-01  2.622e-01  -2.524  0.01161 *
SHORT_RULE_DESCIs Above Auth Limit Ind       -4.451e-01  2.630e-01  -1.693  0.09054 .
SHORT_RULE_DESCIs At DCA Ind                 -1.712e+00  8.400e-01  -2.038  0.04155 *
SHORT_RULE_DESCIs On Black List Ind           4.220e-01  3.035e-01   1.390  0.16443
SHORT_RULE_DESCIs White Credit Balance Ind   -1.228e+00  4.876e-01  -2.519  0.01179 *
SHORT_RULE_DESCIs White Debet Balance Ind     1.033e-01  1.886e-01   0.548  0.58385
SHORT_RULE_DESCIs White No Balance Ind       -1.280e+00  3.139e-01  -4.077 4.55e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 18383  on 316538  degrees of freedom
Residual deviance: 15306  on 316446  degrees of freedom
AIC: 15492

Number of Fisher Scoring iterations: 13
```

6.3 The Reduced Logistic Model

Summary of the reduced logistic model, trained on the original data with no resampling.

```
> summary(model2)

Call:
```

```
glm(formula = ReFinInd ~ ., family = "binomial", data = data2[training,
    ])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.3252  -0.1005  -0.0408  -0.0180   4.4503

Coefficients:
                                                Estimate Std. Error z value Pr(>|z|)
(Intercept)                                   -7.475e+00  1.110e+00  -6.737 1.62e-11 ***
CustomerAge                                   -1.501e-02  2.287e-03  -6.562 5.30e-11 ***
MonthsSinceAccountCreated                     -3.235e-03  6.153e-04  -5.257 1.47e-07 ***
CLOSING_BALANCE_AMT                           -1.318e-05  1.384e-06  -9.520  < 2e-16 ***
HAS_ESTATEMENT_AGREEMENT_IND1                  1.640e-01  5.594e-02   2.931 0.003379 **
average_credit_limit_last12                    2.929e-06  1.082e-06   2.706 0.006804 **
rev_uti_currmth                                3.450e-03  5.108e-04   6.754 1.44e-11 ***
Segment9NameDelinquent                         1.338e+00  1.000e+00   1.337 0.181173
Segment9NameEMOB - Active in last 6 mths       1.539e+00  1.099e+00   1.401 0.161261
Segment9NameEMOB - Not active last 6 mths     -3.312e-03  1.495e+00  -0.002 0.998233
Segment9NameLast active 4-6 mths ago          -1.144e+00  1.486e+00  -0.770 0.441451
Segment9NameLast active 7-12 mths ago         -9.514e-01  1.485e+00  -0.641 0.521783
Segment9NameOccasional Revolver                1.977e+00  1.086e+00   1.821 0.068575 .
Segment9NameRevolved only                      1.214e+00  1.104e+00   1.100 0.271303
Segment9NameRevolver                           1.807e+00  1.085e+00   1.666 0.095668 .
Segment9NameTransactor                         3.589e-02  1.102e+00   0.033 0.974030
Score                                          2.882e-01  2.502e-02  11.517  < 2e-16 ***
SUM_of_CreditLimitIncreaseFlag                 4.024e-01  5.617e-02   7.165 7.79e-13 ***
SUM_of_AIRLINEL12                             -2.579e-05  7.013e-06  -3.677 0.000236 ***
SUM_of_CLOTHING_STORESL12                     -1.510e-05  5.941e-06  -2.542 0.011028 *
SHORT_RULE_DESCDCA 24 To 36 Ind               -5.707e-01  3.973e-01  -1.436 0.150873
SHORT_RULE_DESCExeeded Num Of Dunnings 12 Ind -6.021e-01  3.314e-01  -1.817 0.069255 .
SHORT_RULE_DESCHad Collection Adv 6 To 12 Ind -3.711e-01  2.629e-01  -1.411 0.158115
SHORT_RULE_DESCHad Neg Status 0 to 6 Ind      -1.191e-01  2.010e-01  -0.593 0.553501
SHORT_RULE_DESCHas Balance Above CL Ind        2.652e-01  1.930e-01   1.374 0.169437
SHORT_RULE_DESCHas Missed Payments Ind        -3.839e-01  2.344e-01  -1.637 0.101528
SHORT_RULE_DESCHas Neg Status Ind             -2.026e-01  3.999e-01  -0.507 0.612450
SHORT_RULE_DESCHas Too High Total Limit Ind   -7.124e-01  2.564e-01  -2.778 0.005465 **
SHORT_RULE_DESCIs Above Auth Limit Ind        -4.304e-01  2.612e-01  -1.648 0.099370 .
SHORT_RULE_DESCIs At DCA Ind                  -1.832e+00  8.355e-01  -2.193 0.028311 *
SHORT_RULE_DESCIs On Black List Ind            5.064e-01  3.001e-01   1.687 0.091513 .
SHORT_RULE_DESCIs White Credit Balance Ind    -1.258e+00  4.850e-01  -2.594 0.009496 **
SHORT_RULE_DESCIs White Debet Balance Ind      1.382e-01  1.814e-01   0.762 0.446286
SHORT_RULE_DESCIs White No Balance Ind        -1.409e+00  3.102e-01  -4.542 5.56e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 18383  on 316538  degrees of freedom
Residual deviance: 15436  on 316505  degrees of freedom
AIC: 15504

Number of Fisher Scoring iterations: 12
```

6.4 Random Forest

This is not very informative, but one such print-out should be included.

```
> rf2_220$call
randomForest(formula = ReFinInd ~ ., data = data2[training, ],
    ntree = n, replace = TRUE, importance = TRUE, cutoff = c(219/220, 1/220))
```

Appendix C

The code is given here. The results from sampling are reproducible, and can be reproduced by the two first files, usefulFunctions.R and sampling.R. The code should be run in the order they appear here.

6.5 usefulFunctions.R

```
sigmoid = function(x){
    return(exp(x)/(1+exp(x)))
}

confMatrix = function(prediction, referenceValue){
    y = sum(referenceValue)
    type1 = sum(prediction>referenceValue)
    type2 = sum(prediction<referenceValue)
    truePos = y-type2
    trueNeg = length(prediction)-type1-type2-truePos
    return(c(trueNeg,type1,type2,truePos))
}


# a function that is useful and easy in scripts. creates a confusion matrix
cmFull = function(xTest,coefs,alpha=0.5,yTest){
    return(confMatrix(classify(sigmoid(linPred(xTest,coefs)),alpha),yTest))
}


# the function below takes the probabilities x determined by model from
# an undersampled dataset and makes them unbiased relative to the
# original dataset. p = relative frequency of positive response in the
# original dataset, u = same for undersampled dataset
probUnbiased = function(p,u,x){
    a = (p/u)*x
    b = a + (1-x)*(1-p)/(1-u)
    return(a/b)
}


linPred = function(xTest,coefs){
    xTest = cbind(1,xTest)
    xTest = as.matrix(xTest)
    coefs = as.matrix(coefs)
    return(xTest%*%coefs)
}


baccGen = function(tn,fp,fn,tp,lambda){
    a = tn/(tn+fp)
    b = tp/(fn+tp)
    return((1-lambda)*a+lambda*b)
}


cutOffValue = function(p,u){
    # p = proportion of positive responses in main data,
    # u = proportion in undersampled
    return(u*(1-p)/(u*(1-p)+p*(1-u)))
}


classify = function(probabilities,cutOff=0.5){
    # True = 1, False = 0
    return(as.numeric(probabilities>=cutOff))
}


# warning! when using the two functions below, set mySeed equal for both
smoteDataCont = function(df, numNew, a = 0.5, mySeed){
    n = nrow(df)
    set.seed(mySeed)
    sample1 = sample(n, numNew, replace = TRUE)
    sample2 = sample(n, numNew, replace = TRUE)
    lambdas = runif(numNew, min = a, max = 1)
    smoted = lambdas*df[sample1,]+(1-lambdas)*df[sample2,]
    return(smoted)
}


smoteDataCat = function(df, numNew, mySeed){
    n = nrow(df)
    set.seed(mySeed)
    sample1 = sample(n, numNew, replace = TRUE)
    smoted = df[sample1,]
    return(smoted)
}
```

6.6 sampling.R

```
library(stats)
library(corrplot)

source('~/Skole/Master/usefulFunctionsMaster.R') # desktop
#source('~/Skole/Master/Master/usefulFunctionsMaster.R') # laptop

ratios = c(1,4,9,19,49,99,219,499,999)
cutoffs = 1/(ratios+1)
mu = 1
n1 = 1000

# showing off a mixed distribution
##################################################################
n = 500000
ratio = 9
x = rnorm(ratio*n,mean = -2)
y = rnorm(n,mean = 2)
distribution = c(x,y) # used to show mixed distribution
response = c(rep(0,ratio*n),rep(1,n))
##################################################################


#creating table "logregImba"
##################################################################
mu = 1
n1 = 1000
sigma = 1

set.seed(2019)
xList = list()
xTestList = list()
yList = list()
dfList = list()
coefs = list()
cmList = list()
accuracyList = list()
for(i in 1:length(ratios)){
    xList[[i]] = c(rnorm(n1*ratios[i], mean = -mu),rnorm(n1, mean = mu))
    xTestList[[i]] = c(rnorm(n1*ratios[i], mean = -mu),rnorm(n1, mean = mu))
    yList[[i]] = c(rep(0,n1*ratios[i]),rep(1,n1))
    dfList[[i]] = data.frame(xList[[i]],yList[[i]])
    colnames(dfList[[i]]) = c("x","y")
    coefs[[i]] = glm(y ~ x, data = dfList[[i]], family = "binomial")$coefficients
    cmList[[i]] = cmFull(xTestList[[i]],coefs[[i]],0.5,yList[[i]])
    accuracyList[[i]] = (cmList[[i]][1]+cmList[[i]][4])/sum(cmList[[i]])
}
accuracyList

#coefs # these are found in the table
xCutOff = 0
for(i in 1:length(ratios)){
    xCutOff[i] = -coefs[[i]][1]/coefs[[i]][2]
}
xCutOff
pnorm(mu+xCutOff) # true negative rate
pnorm(xCutOff-mu, lower.tail = FALSE) # true positive rate
(pnorm(mu+xCutOff) + pnorm(xCutOff-mu, lower.tail = FALSE))/2

summary(glm(y~x, data = dfList[[1]], family ="binomial"))
##################################################################


set.seed(1337)
# creating table "cutoffs"
##################################################################
xList = list()
```

```
xTest = list()
yList = list()
dfList = list()
coefs = list()
cmList = list()
for(i in 1:length(ratios)){
    xList[[i]] = c(rnorm(n1*ratios[i], mean = -mu),rnorm(n1, mean = mu))
    xTest[[i]] = c(rnorm(n1*ratios[i], mean = -mu),rnorm(n1, mean = mu))
    yList[[i]] = c(rep(0,n1*ratios[i]),rep(1,n1))
    dfList[[i]] = data.frame(xList[[i]],yList[[i]])
    colnames(dfList[[i]]) = c("x","y")
    coefs[[i]] = glm(y ~ x, data = dfList[[i]], family = "binomial")$coefficients
    cmList[[i]] = cmFull(xTest[[i]],coefs[[i]],0.5,yList[[i]]) # this line is unnecessary
}

cmList = list()
baccList = list()
for(i in 1:length(ratios)){
    baccList[[i]] = list()
    for(j in 1:length(cutoffs)){
        cmList[[j]] = cmFull(xTest[[i]],coefs[[i]],cutoffs[j],yList[[i]])
        temp = cmList[[j]]
        baccList[[i]][j] = baccGen(temp[1],temp[2],temp[3],temp[4],0.5)
    }
}


##################################################################


# sampling, categorical (kind of)
set.seed(12345)
# this section was a failed attempt and has been left out of the
# theory and results
##################################################################
n1 = 1000
alfa0 = 0
alfa1 = 0
beta0 = 0.1
beta1 = 0.1

xList = list()
xTest = list()
yList = list()
yTest = list()
yTest2 = list()

dfList = list()
coefs = list()
cmList = list()
accuracyList = list()
baccList = list()
for(i in 1:length(ratios)){
    cc = 0.2*log(ratios[i])
    x0 = -runif(n1*ratios[i],alfa0+cc,beta0+cc)
    x1 = runif(n1,alfa1,beta1)
    xList[[i]] = c(x0,x1)

    x0 = -runif(n1*ratios[i],alfa0+cc,beta0+cc)
    x1 = runif(n1,alfa1,beta1)
    xTest[[i]] = c(x0,x1)
    u1 = runif(n1*ratios[i]+n1)
    u2 = runif(n1*ratios[i]+n1)

    yList[[i]] = as.numeric(u1<=sigmoid(xList[[i]]))
    yTest[[i]] = as.numeric(u2<=sigmoid(xTest[[i]]))
    yTest2[[i]] = c(rep(0,n1*ratios[i]),rep(1,n1))

    dfList[[i]] = data.frame(xList[[i]],yList[[i]])

    colnames(dfList[[i]]) = c("x","y")
    coefs[[i]] = glm(y ~ x, data = dfList[[i]], family = "binomial")$coefficients
    cmList[[i]] = cmFull(xTest[[i]],coefs[[i]],0.5,yTest2[[i]])

    accuracyList[[i]] = (cmList[[i]][1]+cmList[[i]][4])/sum(cmList[[i]])
    baccList[[i]] = 0.5*cmList[[i]][1]/(cmList[[i]][1]+cmList[[i]][2])
    baccList[[i]] = baccList[[i]]+0.5*cmList[[i]][4]/(cmList[[i]][3]+cmList[[i]][4])
}


rm(xList,xTest,yList,yTest,yTest2,dfList) # comment this out of if u dont have RAM issues
rm(u1,u2,x0,x1) # same as above

cmList
accuracyList
baccList
# this section was a failure and has not been included in the results or theory
##################################################################
```

```
n1 = 20
set.seed(n1)
# we set n1 = 20 and train models. check to see what works better
# of oversampling and undersampling
# switch n1 to 1000 after producing results for n1 = 20 and repeat
#################################################################
ratio = 219 # same as in the main data set by sparebank 1
mu = 1
limit1 = n1*ratio-n1+1
limit2 = n1*ratio+n1
numNew = (ratio-1)*n1

xTest = c(rnorm(10000, mean = -mu),rnorm(10000, mean = mu))
yTest = c(rep(0,10000), rep(1,10000))

coefList = list()
coefUnder = list()
coefOver = list()
coefSmote = list()

cmList = list()
cmUnder = list()
cmOver = list()
cmSmote = list()
baccList = list()

for(i in 1:10){
    x = c(rnorm(n1*ratio, mean = -mu),rnorm(n1, mean = mu))
    y = c(rep(0,n1*ratio), rep(1,n1))
    df = data.frame(x,y)
    ind1 = seq((n1*ratio+1),(n1*ratio+n1))
    mySample = sample(ind1, (ratio-1)*n1, replace = TRUE)

    coefList[[i]] = glm(y ~ x, data = df, family = "binomial")$coefficients
    coefUnder[[i]] = glm(y ~ x, data = df[limit1:limit2,], family = "binomial")$coefficients
    dfOver = rbind(df,df[mySample,])
    coefOver[[i]] = glm(y ~ x, data = dfOver, family = "binomial")$coefficients

    #smote = smoteDataCont(df[4381:4400,],numNew,0.5,i)
    smote = smoteDataCont(df[(ratio*n1+1):((1+ratio)*n1),],numNew,0.5,i)
    dfSmote = rbind(df,smote)
    coefSmote[[i]] = glm(y ~ x, data = dfSmote, family = "binomial")$coefficients

    #print(-coefList[[i]][1]/coefList[[i]][2])
    cmList[[i]] = cmFull(xTest,coefList[[i]],0.5,yTest)
    cmUnder[[i]] = cmFull(xTest,coefUnder[[i]],0.5,yTest)
    cmOver[[i]] = cmFull(xTest,coefOver[[i]],0.5,yTest)
    cmSmote[[i]] = cmFull(xTest,coefSmote[[i]],0.5,yTest)
}

beta0Under = c()
beta1Under = c()
beta0Over = c()
beta1Over = c()
beta0Smote = c()
beta1Smote = c()


for(i in 1:10){
    beta0Under = c(beta0Under,coefUnder[[i]][1])
    beta1Under = c(beta1Under,coefUnder[[i]][2])
    beta0Over = c(beta0Over,coefOver[[i]][1])
    beta1Over = c(beta1Over,coefOver[[i]][2])
    beta0Smote = c(beta0Smote,coefSmote[[i]][1])
    beta1Smote = c(beta1Smote,coefSmote[[i]][2])
}
x = 0
for(i in 1:10){
    print(0.5*cmUnder[[i]][1]+0.5*cmUnder[[i]][4])
    x = x + 0.5*cmUnder[[i]][1]+0.5*cmUnder[[i]][4]
}
x
x = 0
for(i in 1:10){
    print(0.5*cmOver[[i]][1]+0.5*cmOver[[i]][4])
    x = x + 0.5*cmOver[[i]][1]+0.5*cmOver[[i]][4]
}
x
x = 0
tnr = 0
tpr = 0
for(i in 1:10){
    print(0.5*cmSmote[[i]][1]+0.5*cmSmote[[i]][4])
    x = x + 0.5*cmSmote[[i]][1]+0.5*cmSmote[[i]][4]
    tnr = tnr + cmSmote[[i]][1]
    tpr = tpr + cmSmote[[i]][4]
}
x
tnr/100000
tpr/100000
```

```
x = 0
for(i in 1:10){
    print(0.5*cmList[[i]][1]+0.5*cmList[[i]][4])
    x = x + 0.5*cmList[[i]][1]+0.5*cmList[[i]][4]
}
x

#################################################################


# finds the fraction of SMOTEd observations with x less than 0
#################################################################
set.seed(2019)
x = rnorm(10000,mean = 1)
y = rep(1, 10000)
mydf = data.frame(x,y)
ss = smoteDataCont(mydf,1000000,0.5,2019)
# sum(ss$x<0) # 114968, so roughly 11.5 %
#################################################################
```

6.7 dataPrep.R

```
library(leaps)
library(haven)
library(dplyr)
library(gsubfn)
library(MASS)


#dataset = read_sas("Skole/Master/Master/Data/yarefin_trainset_small201810 ok.sas7bdat") # laptop
dataset = read_sas("Skole/Master/yarefin_trainset_small201810 ok.sas7bdat") # desktop
datasetOriginal = dataset
save(datasetOriginal, file = "datasetOriginal.RData")
rm(datasetOriginal)

# diagnostics
###########################################################################
# sum(colSums(is.na(dataset))) # 2 357 690
# ncol(dataset) # 78
# nrow(dataset) # 614465
# sum(dataset$ReFinInd==1) # 2665

###########################################################################

acc_id = dataset$BK_ACCOUNT_ID
acc_id = sort(acc_id)
dataset = dataset[match(acc_id,dataset$BK_ACCOUNT_ID),]
rm(acc_id)
dataset = subset(dataset,
                 select = -c(lead1YearMonth,lead2YearMonth,
                             lead3YearMonth,Segment23Name,
                             PeriodId,Date,YearMonth))
count = 0
mylist = c(0)
for(i in 2:length(dataset$BK_ACCOUNT_ID)){
    if(dataset$BK_ACCOUNT_ID[i]==dataset$BK_ACCOUNT_ID[i-1]){
        count = count + 1
        mylist = c(mylist,i)
    }
}
# count # = 3700

mylist=mylist[2:length(mylist)]
dataset = dataset[-mylist,]
dataset[is.na(dataset)] = 0

dataset = dataset[which(!dataset$CustomerAge==0),]
#sum(dataset$ReFinInd==1) # 2129
dataset = subset(dataset, select = -BK_ACCOUNT_ID)
dataset$ReFinInd = as.factor(dataset$ReFinInd)
dataset$PRODUCT_NAME = as.factor(dataset$PRODUCT_NAME)
dataset$STATEMENT_DUE_DAY_OF_MONTH_NUM = as.factor(dataset$STATEMENT_DUE_DAY_OF_MONTH_NUM)
dataset$ApplicationSalesChannel = as.factor(dataset$ApplicationSalesChannel)
dataset$CAMPAIGN_NAME = as.factor(dataset$CAMPAIGN_NAME)
dataset$DISTRIBUTOR_NAME = as.factor(dataset$DISTRIBUTOR_NAME)
dataset$GENDER_NAME = as.factor(dataset$GENDER_NAME)
dataset$Segment9Name = as.factor(dataset$Segment9Name)
dataset$SHORT_RULE_DESC = as.factor(dataset$SHORT_RULE_DESC)
dataset$PNRSerial = floor(dataset$PNRSerial/10)
dataset$PNRSerial = as.factor(dataset$PNRSerial)
dataset$HAS_DIRECT_DEBIT_AGREEMENT_IND = as.factor(dataset$HAS_DIRECT_DEBIT_AGREEMENT_IND)
dataset$HAS_ESTATEMENT_AGREEMENT_IND = as.factor(dataset$HAS_ESTATEMENT_AGREEMENT_IND)
```

```
indices = which(dataset$Segment9Name=="Not active in last 12 mths")
# sum(dataset[indices,]$ReFinInd == 1) # = 9, out of 144 199
dataset = dataset[-indices,] # removing accounts that have not been active for 12 months
dataset$Segment9Name = factor(dataset$Segment9Name) # removes non-existent factor levels

#############################################################################
set.seed(2019)
indices = sample(466539,40000, replace = FALSE)


#for the "dataset" section
############################################################################
# dataSmall = dataset[indices,]
# modelUseless = glm(ReFinInd ~ ., data = dataSmall, family = "binomial")
# coefUseless = modelUseless$coefficients
#
# summary(modelUseless)
#this is the model that shows collinearity in
#188 parameters + intercept
#############################################################################
dataset = subset(dataset,
                 select = -c(CAMPAIGN_NAME, PNRSerial,DISTRIBUTOR_NAME, PRODUCT_NAME))

# sum(dataset$ReFinInd==0) # 464 419
sum(colSums(dataset==0)) # 18 701 432


save(dataset, file = "dataset.RData")
#############################################################################

dataCont = subset(dataset,
                  select = -c(SHORT_RULE_DESC,Segment9Name,
                              GENDER_NAME,ApplicationSalesChannel,
                              STATEMENT_DUE_DAY_OF_MONTH_NUM,
                              HAS_DIRECT_DEBIT_AGREEMENT_IND,
                              HAS_ESTATEMENT_AGREEMENT_IND,ReFinInd))
dataCat = subset(dataset,
                 select =  c(SHORT_RULE_DESC,Segment9Name,
                             GENDER_NAME,ApplicationSalesChannel,
                             STATEMENT_DUE_DAY_OF_MONTH_NUM,
                             HAS_DIRECT_DEBIT_AGREEMENT_IND,
                             HAS_ESTATEMENT_AGREEMENT_IND,ReFinInd))
save(dataCont, file = "dataCont.RData")
save(dataCat, file = "dataCat.RData")
#############################################################################

set.seed(1000)
# stratified sampling, creating training and test set
# test set will have 150 000 observations, with 682 positive observations
##################################################################
indices0 = which(dataset$ReFinInd==0)
indices1 = which(dataset$ReFinInd==1)
#sum(dataset$ReFinInd==1) 2120

# 2120*(n-150000)/n # = 1438.385
# n-150000-1438 # = 315101
train1 = sample(indices1, 1438, replace = FALSE) # stratified sampling
train0 = sample(indices0, 315101, replace = FALSE) # stratified sampling
training = c(train1,train0) # indices of training set
##################################################################


# model selection
#############################################################################
numPredictors = seq(2,93)
regfit = regsubsets(ReFinInd ~ ., data = dataset[training,], nvmax = 93, method = "forward")
regfitsummary = summary(regfit)

aic = regfitsummary$bic + (2-log(dim(dataset[training,])[1]))*numPredictors
nVarAIC = which.min(aic)
nVarBIC = which.min(regfitsummary$bic)


varSmall = which(regfitsummary$which[nVarBIC,])
varSmall # variables in data2 below
varLarge = which(regfitsummary$which[nVarAIC,])
varLarge

# variables below found from forward selection
data2 = subset(dataset,
               select = c(CustomerAge,MonthsSinceAccountCreated,
                          CLOSING_BALANCE_AMT,HAS_ESTATEMENT_AGREEMENT_IND,
                          average_credit_limit_last12,rev_uti_currmth,
                          Segment9Name,Score,SUM_of_CreditLimitIncreaseFlag,
                          SUM_of_AIRLINEL12,SUM_of_CLOTHING_STORESL12,
                          SHORT_RULE_DESC, ReFinInd))
save(data2, file = "data2.RData")


data2Cat = subset(data2,
                  select = c(HAS_ESTATEMENT_AGREEMENT_IND,
                             Segment9Name,SHORT_RULE_DESC,ReFinInd))
data2Cont = subset(data2,
                  select = -c(HAS_ESTATEMENT_AGREEMENT_IND,
                              Segment9Name,SHORT_RULE_DESC,ReFinInd))
save(data2Cat, file = "data2Cat.RData")
save(data2Cont, file = "data2Cont.RData")


############################################################################
```

6.8 logReg.R

```
ratios = c(1,4,9,19,49,99,219,499,999)
cutoffs = 1/(ratios+1)
references = as.numeric(dataset$ReFinInd)-1



# model is trained and saved here
##################################################################
model1 = glm(ReFinInd ~ ., data = dataset[training,], family = "binomial") # full model
save(model1, file = "model1.rda")
model2 = glm(ReFinInd ~ ., data = data2[training,], family = "binomial") # reduced model
save(model2, file = "model2.rda")

##################################################################


# full model, varying cut off
# also, reduced model
##################################################################
nTest = 150000
cmList = list()
baccList = list()
accList = list()

cmList2 = list()
baccList2 = list()
accList2 = list()

references = as.numeric(dataset$ReFinInd)-1
for(i in 1:length(cutoffs)){
    cmList[[i]] = confMatrix(classify(sigmoid(predict(model1,dataset[-training,])),
                                      cutoffs[i]),references[-training])
    temp = cmList[[i]]
    baccList[[i]] = baccGen(temp[1],temp[2],temp[3],temp[4],0.5)
    accList[[i]] = (temp[1]+temp[4])/nTest

    cmList2[[i]] = confMatrix(classify(sigmoid(predict(model2,data2[-training,])),
                                      cutoffs[i]),references[-training])
    temp = cmList2[[i]]
    baccList2[[i]] = baccGen(temp[1],temp[2],temp[3],temp[4],0.5)
    accList2[[i]] = (temp[1]+temp[4])/nTest


}
cmList
baccList
accList
cmList2
baccList2
accList2
##################################################################


# smote
# this section is commented out so i dont accidentally run it again
##################################################################
imba = c(0.05,0.1,0.2) # desired training set imbalance
# achieved by under- and oversampling and SMOTE
numNew = round((imba*316539-1438)/(1-imba)) # number of new oversampled/
# smoted observations to create the desired imbalance (balance)


mySeed = 1995
dataCatSmoted = smoteDataCat(dataCat[train1,],numNew[1],mySeed)
dataContSmoted = smoteDataCont(dataCont[train1,],numNew[1],0.5,mySeed)
dataSmoted05 = cbind(dataContSmoted,dataCatSmoted)
dataSmoted05 = rbind(dataSmoted05,dataset[training,])
save(dataSmoted05, file = "dSmote05.rda")

lmSmote05 = glm(data = dataSmoted05, ReFinInd ~ ., family = "binomial")
```

```
save(lmSmote05, file = "lmSmote05.rda")
rm(lmSmote05)

mySeed = 1996
dataCatSmoted = smoteDataCat(dataCat[train1,],numNew[2],mySeed)
dataContSmoted = smoteDataCont(dataCont[train1,],numNew[2],0.5,mySeed)
dataSmoted10 = cbind(dataContSmoted,dataCatSmoted)
dataSmoted10 = rbind(dataSmoted10,dataset[training,])
save(dataSmoted10, file = "dSmote10.rda")

lmSmote10 = glm(data = dataSmoted10, ReFinInd ~ ., family = "binomial")
save(lmSmote10, file = "lmSmote10.rda")
rm(lmSmote10)

mySeed = 1997
dataCatSmoted = smoteDataCat(dataCat[train1,],numNew[3],mySeed)
dataContSmoted = smoteDataCont(dataCont[train1,],numNew[3],0.5,mySeed)
dataSmoted20 = cbind(dataContSmoted,dataCatSmoted)
dataSmoted20 = rbind(dataSmoted20,dataset[training,])
save(dataSmoted20, file = "dSmote20.rda")

lmSmote20 = glm(data = dataSmoted20, ReFinInd ~ ., family = "binomial")
save(lmSmote20, file = "lmSmote20.rda")
rm(lmSmote20)

##################################################################

# oversampling
# this section is commented out so i dont accidentally run it again
##################################################################
imba = c(0.05,0.1,0.2) # desired training set imbalance
# achieved by under- and oversampling and SMOTE
numNew = round((imba*316539-1438)/(1-imba)) # number of new oversampled/
# smoted observations to create the desired imbalance (balance)

set.seed(3000)
newSamples = list()
overSampled = list()
for(i in 1:3){
    newSamples[[i]] = sample(train1,numNew[[i]], replace = TRUE)
    overSampled[[i]] = c(newSamples[[i]],training)
}

lmOver05 = glm(ReFinInd ~ ., data = dataset[overSampled[[1]],], family = "binomial")
save(lmOver05, file = "lmOver05.rda")
rm(lmOver05)

lmOver10 = glm(ReFinInd ~ ., data = dataset[overSampled[[2]],], family = "binomial")
save(lmOver10, file = "lmOver10.rda")
rm(lmOver10)

lmOver20 = glm(ReFinInd ~ ., data = dataset[overSampled[[3]],], family = "binomial")
save(lmOver20, file = "lmOver20.rda")
rm(lmOver20)

##################################################################

#undersampling
set.seed(1111)
# if set.seed is set to 4000, then lmUnder20 cant predict the dataset
# as application sales channel has unseen levels
##################################################################
imba = c(0.05,0.1,0.2)
ratios = c(19,9,4)
n1 = 1438

samples = list()
underSampled = list()
for(i in 1:3){
    samples[[i]] = sample(train0, ratios[i]*n1, replace = FALSE)
    underSampled[[i]] = c(train1,samples[[i]])
}

lmUnder05 = glm(ReFinInd ~ ., data = dataset[underSampled[[1]],], family = "binomial")
lmUnder10 = glm(ReFinInd ~ ., data = dataset[underSampled[[2]],], family = "binomial")
lmUnder20 = glm(ReFinInd ~ ., data = dataset[underSampled[[3]],], family = "binomial")

##################################################################


# comparing smote, undersampling and oversampling at degree 5,10,20%
# run the section 3 times
##################################################################
cutoffs2 = seq(0.1,0.2,0.005) # this has been changed a few times

baccUnder = c()
baccOver = c()
baccSmote = c()

# need to change the names of the models to test all 9
```

```
for(i in 1:length(cutoffs2)){
    temp = confMatrix(classify(sigmoid(predict(lmUnder20,dataset[-training,])),
                               cutoffs2[i]),references[-training])
    baccUnder = c(baccUnder,baccGen(temp[1],temp[2],temp[3],temp[4],0.5))

    temp = confMatrix(classify(sigmoid(predict(lmOver20,dataset[-training,])),
                               cutoffs2[i]),references[-training])
    baccOver = c(baccOver,baccGen(temp[1],temp[2],temp[3],temp[4],0.5))

    temp = confMatrix(classify(sigmoid(predict(lmSmote20,dataset[-training,])),
                               cutoffs2[i]),references[-training])
    baccSmote = c(baccSmote,baccGen(temp[1],temp[2],temp[3],temp[4],0.5))
}
max(baccSmote)
max(baccUnder)
max(baccOver)
##################################################################
```

6.9 randomForests.R

```
library(randomForest)
start.time = Sys.time()

references = as.numeric(dataset$ReFinInd)-1

ratios = c(1,4,9,19,49,99,219,499,999)
cutoffs = 1/(ratios+1)

set.seed(2019)
n = 200
##################################################################
rf_50 = randomForest(ReFinInd ~ ., data = dataset[training,], ntree = n,
                  replace = TRUE, importance = TRUE, cutoff = c(0.5,0.5))
save(rf_50, file = "rf_50.RData")
rm(rf_50)


rf_10 = randomForest(ReFinInd ~ ., data = dataset[training,], ntree = n,
                     replace = TRUE, importance = TRUE, cutoff = c(0.9,0.1))
save(rf_10, file = "rf_10.RData")
rm(rf_10)


rf_02 = randomForest(ReFinInd ~ ., data = dataset[training,], ntree = n,
                     replace = TRUE, importance = TRUE, cutoff = c(0.98,0.02))
save(rf_02, file = "rf_02.RData")
rm(rf_02)


rf_220 = randomForest(ReFinInd ~ ., data = dataset[training,], ntree = n,
                     replace = TRUE, importance = TRUE, cutoff = c(219/220,1/220))
save(rf_220, file = "rf_220.RData")
rm(rf_220)
##################################################################

end.time = Sys.time()
end.time - start.time


set.seed(2020)
n = 200
##################################################################
rf2_50 = randomForest(ReFinInd ~ ., data = data2[training,], ntree = n,
                     replace = TRUE, importance = TRUE, cutoff = c(0.5,0.5))
save(rf2_50, file = "rf2_50.RData")
rm(rf2_50)


rf2_10 = randomForest(ReFinInd ~ ., data = data2[training,], ntree = n,
                     replace = TRUE, importance = TRUE, cutoff = c(0.9,0.1))
save(rf2_10, file = "rf2_10.RData")
rm(rf2_10)


rf2_02 = randomForest(ReFinInd ~ ., data = data2[training,], ntree = n,
                     replace = TRUE, importance = TRUE, cutoff = c(0.98,0.02))
save(rf2_02, file = "rf2_02.RData")
rm(rf2_02)


rf2_220 = randomForest(ReFinInd ~ ., data = data2[training,], ntree = n,
                      replace = TRUE, importance = TRUE, cutoff = c(219/220,1/220))
save(rf2_220, file = "rf2_220.RData")
```

```
rm(rf2_220)
##################################################################




#smote
##################################################################
data2Smoted05 = subset(dataSmoted05,
                    select = c(CustomerAge,MonthsSinceAccountCreated,
                          CLOSING_BALANCE_AMT,HAS_ESTATEMENT_AGREEMENT_IND,
                          average_credit_limit_last12,rev_uti_currmth,
                          Segment9Name,Score,SUM_of_CreditLimitIncreaseFlag,
                          SUM_of_AIRLINEL12,SUM_of_CLOTHING_STORESL12,
                          SHORT_RULE_DESC, ReFinInd))

data2Smoted10 = subset(dataSmoted10,
                       select = c(CustomerAge,MonthsSinceAccountCreated,
                                  CLOSING_BALANCE_AMT,HAS_ESTATEMENT_AGREEMENT_IND,
                                  average_credit_limit_last12,rev_uti_currmth,
                                  Segment9Name,Score,SUM_of_CreditLimitIncreaseFlag,
                                  SUM_of_AIRLINEL12,SUM_of_CLOTHING_STORESL12,
                                  SHORT_RULE_DESC, ReFinInd))

data2Smoted20 = subset(dataSmoted20,
                       select = c(CustomerAge,MonthsSinceAccountCreated,
                                  CLOSING_BALANCE_AMT,HAS_ESTATEMENT_AGREEMENT_IND,
                                  average_credit_limit_last12,rev_uti_currmth,
                                  Segment9Name,Score,SUM_of_CreditLimitIncreaseFlag,
                                  SUM_of_AIRLINEL12,SUM_of_CLOTHING_STORESL12,
                                  SHORT_RULE_DESC, ReFinInd))

##################################################################


#oversampling
##################################################################
imba = c(0.05,0.1,0.2) # desired training set imbalance
# achieved by under- and oversampling and SMOTE
numNew = round((imba*316539-1438)/(1-imba)) # number of new oversampled/
# smoted observations to create the desired imbalance (balance)

set.seed(3000)
newSamples = list()
overSampled = list()
for(i in 1:3){
    newSamples[[i]] = sample(train1,numNew[[i]], replace = TRUE)
    overSampled[[i]] = c(newSamples[[i]],training)
}
##################################################################


#undersampling
set.seed(1111)
# if set.seed is set to 4000, then lmUnder20 cant predict the dataset
# as application sales channel has unseen levels
##################################################################
imba = c(0.05,0.1,0.2)
ratios = c(19,9,4)
n1 = 1438

samples = list()
underSampled = list()
for(i in 1:3){
    samples[[i]] = sample(train0, ratios[i]*n1, replace = FALSE)
    underSampled[[i]] = c(train1,samples[[i]])
}
##################################################################

start.time = Sys.time()

# training 9 models, resampling at 3 levels, 3 methods
n = 200
##################################################################
rf2_Over05 = randomForest(ReFinInd ~ ., data = data2[overSampled[[1]],], ntree = n,
                      replace = TRUE, importance = TRUE, cutoff = c(0.95,0.05))
rf2_Over10 = randomForest(ReFinInd ~ ., data = data2[overSampled[[2]],], ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.90,0.10))
rf2_Over20 = randomForest(ReFinInd ~ ., data = data2[overSampled[[3]],], ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.80,0.20))
save(rf2_Over05, file = "rf2_Over05.rda")
save(rf2_Over10, file = "rf2_Over10.rda")
save(rf2_Over20, file = "rf2_Over20.rda")
rm(rf2_Over20,rf2_Over10,rf2_Over05)

rf2_Under05 = randomForest(ReFinInd ~ ., data = data2[underSampled[[1]],], ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.95,0.05))
rf2_Under10 = randomForest(ReFinInd ~ ., data = data2[underSampled[[2]],], ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.90,0.10))
```

```
rf2_Under20 = randomForest(ReFinInd ~ ., data = data2[underSampled[[3]],], ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.80,0.20))
save(rf2_Under05, file = "rf2_Under05.rda")
save(rf2_Under10, file = "rf2_Under10.rda")
save(rf2_Under20, file = "rf2_Under20.rda")
rm(rf2_Under20,rf2_Under10,rf2_Under05)

rf2_Smote05 = randomForest(ReFinInd ~ ., data = data2Smoted05, ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.95,0.05))
rf2_Smote10 = randomForest(ReFinInd ~ ., data = data2Smoted10, ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.90,0.10))
rf2_Smote20 = randomForest(ReFinInd ~ ., data = data2Smoted20, ntree = n,
                          replace = TRUE, importance = TRUE, cutoff = c(0.80,0.20))
save(rf2_Smote05, file = "rf2_Smote05.rda")
save(rf2_Smote10, file = "rf2_Smote10.rda")
save(rf2_Smote20, file = "rf2_Smote20.rda")
rm(rf2_Smote20,rf2_Smote10,rf2_Smote05)

##################################################################

end.time = Sys.time()
end.time - start.time


set.seed(7) #yes, the seed actually matters
pp = predict(rf2_Under05,data2[-training,])
table(references[-training],pp)
pp = predict(rf2_Under10,data2[-training,])
table(references[-training],pp)
pp = predict(rf2_Under20,data2[-training,])
table(references[-training],pp)

pp = predict(rf2_Over05,data2[-training,])
table(references[-training],pp)
pp = predict(rf2_Over10,data2[-training,])
table(references[-training],pp)
pp = predict(rf2_Over20,data2[-training,])
table(references[-training],pp)

pp = predict(rf2_Smote05,data2[-training,])
table(references[-training],pp)
pp = predict(rf2_Smote10,data2[-training,])
table(references[-training],pp)
pp = predict(rf2_Smote20,data2[-training,])
table(references[-training],pp)
```