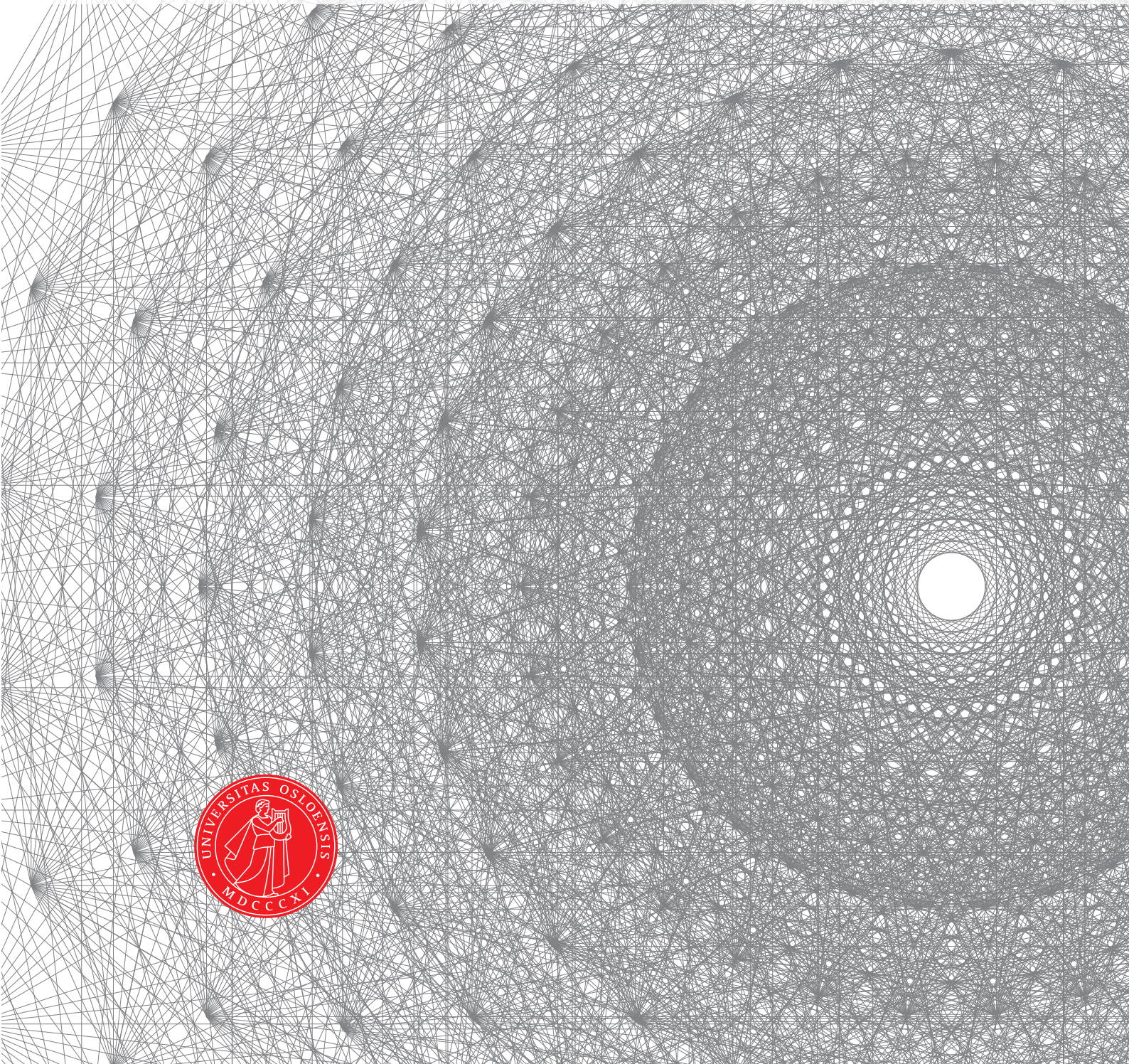
UiO : Department of Mathematics
University of Oslo

Addressing collinearity and class imbalance in logistic regression for statistical fraud detection

Eirik Lødøen Halsteinslid

Master's Thesis, Spring 2019



This master's thesis is submitted under the master's programme *Modelling and Data Analysis*, with programme option *Finance, Insurance and Risk*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Abstract

We study how one can improve upon a logistic regression model for statistical fraud detection. Fraud data are often characterized by uneven class distributions as well as high dependence among covariates. With a focus on recreating such dependence structures found in fraud data, we propose a stochastic model from which we can generate data. The model utilizes copulas to create a highly flexible framework for generating dependent covariates. This allows for a wide range of dependence structures among the covariates, and does not put any restrictions on the marginal distributions for the covariates themselves. We use this data generation scheme to conduct a simulation study of which regularization methods for logistic regression are best suited when covariates are highly dependent. We evaluate this in terms of both prediction and variable selection. The second problem, namely an uneven class distribution, introduces challenges as well. First, selection of an appropriate measure of predictive performance is important. Secondly, it has been demonstrated that some methods may struggle with poor predictive performance on the under-represented class. We study how such a class imbalance affects the predictive performance and variable selection capabilities of the penalized logistic regression methods. In the last part of this thesis we model tax fraud on a real-life data set provided by The Norwegian Tax Administration. Our results show that penalized logistic regression can be a helpful tool for detecting tax fraud.

Acknowledgements

First and foremost, I want to thank my supervisors Ingrid Hobæk Haff and Ingrid Kristine Glad for their guidance and assistance. I also want to thank The Norwegian Computing Center and The Norwegian Tax Administration for providing the data set used in this thesis. A big thank you should also be given to Simon and Vegard for interesting discussions during my work on this thesis. The latter also for letting me borrow his copy of *The Elements of Statistical Learning* when needed. Lastly, I want to thank my family for their help and support during my studies.

Contents

Chapter 1

Introduction

All businesses in Norway must report their Value Added Tax (VAT) paid every two months. It can be an arduous process for a business to produce such reports, since there are many laws and clauses one needs to be familiar with if this is to be done correctly. Thus, errors are occasionally made, and it is in The Norwegian Tax Administration's best interest to find anomalies in VAT reports. Besides detecting and correcting honest mistakes, one is also interested in detecting fraud. The Norwegian Tax Administration routinely carries out controls of VAT reports, and has for this purpose defined several rules and filters which will tag a VAT report as possibly anomalous. This gives the controllers a starting point of which reports to look further into, but the issue with the current approach is that the number of tagged reports far exceeds what one could hope to control. One does not have the time to go through all tagged reports, since each control requires substantial amounts of time and manpower. In this thesis we will focus on a particular data set produced by The Norwegian Tax Administration containing information on previous such VAT controls. For each control we are given a number of attributes regarding both the specific control, and background information about the business being controlled. The data set consists of 50255 actual controls, such that the outcome of each control is known. Each of these controls has a total number of 556 attributes of which 539 are numerical and 17 are categorical. Using these historical data we can build statistical models to gain insight into the underlying mechanisms making a business either to make mistakes in their reporting, or consciously swindle on their reports. Equally important is prediction. That is, we wish to apply our statistical model to new cases in order to estimate their probability of being anomalous. Such models can then be used to help controllers decide which reports one should investigate in the future, thereby making the anomaly detection procedure more effective. Thus our goal is not to perfectly predict

		Predicted	
		Anomalous	Not anomalous
Actual	Anomalous	gain	loss_1
	Not anomalous	loss_2	0

Table 1.1: Table illustrating the loss or gain in the four possible outcomes of a control.

which cases can be anomalous and which cases are not. The final decision about whether or not to investigate a case should be made by controllers in order to utilize their experience and expertise. As the basis for such decision making, we have the four possible outcomes of any control, represented by Table 1.1. The possible gain by uncovering an anomaly, or the potential loss loss_1 caused by not investigating an anomalous case, or loss_2 if one investigates a non-anomalous case will vary. Our goal is to provide a probability such that one can weigh the expected costs of either investigating, or not investigating up against one another. In Chapter 2 we will present the statistical framework necessary to produce such statistical models. We will cover the logistic regression model, and discuss how one should estimate and evaluate these models both generally and specific to the current problem.

If a given report is assigned a high probability of being anomalous, the controllers are also interested in knowing why. This is because the controllers want to be sure that a given case will be worthwhile investigating and feel confident that the case at hand is worth spending valuable time and resources on. However, the large degree of dependence between attributes can make it difficult to provide such information. We essentially run into an ever-present issue in statistical modeling; how one separates correlation from causality. If two attributes A and B are correlated, it can be difficult to separate the effects of A from the effects of B, and to draw a conclusion about which of these attributes truly affect the probability of a report being anomalous. The unfortunate effect on variable selection caused by high dependence among attributes has previously been pointed out as a problem in fraud detection in Løland et al. (2017). The problem of dependence among the attributes in the VAT data set is illustrated in Figure 1.1. We see that the four variables x_2, x_3, x_4 and x_5 are all highly correlated with the variable x_1 (variables names are not provided due to anonymity). Uncovering a causal relationship between these five variables and VAT fraud is difficult, and thus we risk falsely identifying for instance x_1 as a relevant factor for uncovering VAT fraud, when in reality it may be any one (or more) of the other four

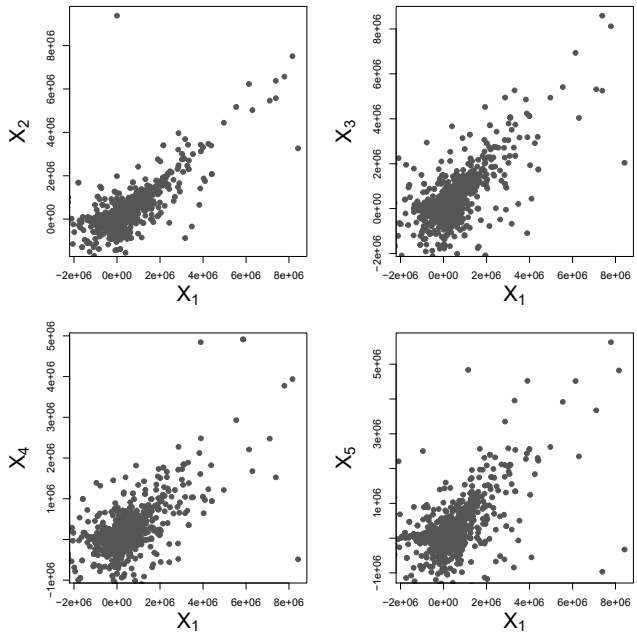


Figure 1.1: Four plots illustrating the degree of dependence between the covariates in the VAT data set.

variables. Methods for selecting attributes in the logistic regression model will be discussed in Chapter 3.

One commonality of fraud data is the low rate at which fraud is perpetrated. This of course varies depending on the specific situation, but a ratio of fraudulent to non-fraudulent incidents of 1 to 100 is not uncommon, and may in some instances be much lower, Bolton and Hand (2002). This has unfortunate effects both when estimating a statistical model and evaluating predictive performance. So called re-sampling methods to remedy problems related to model training in such situations have been proposed, and some of the most used methods are presented in Chapter 4.

Before we study the methods from Chapters 3 and 4 on the real VAT data set we will study these methods in a series of simulation experiments in a controlled environment where we know the true model. We can thus see which methods are best both in terms of prediction and selection of attributes when the covariates are highly dependent and classes unbalanced. The end goal of this is to extrapolate the knowledge we obtain from these simulation experiments to the problem of VAT anomaly detection. How-

ever, it is necessary that the properties of the data sets we generate in our simulations resemble those of the true data set. To this end, we perform a preliminary analysis of the VAT data set, and based on these results construct a stochastic model from which we can sample new observations in Chapter 5. We focus particularly on recreating the dependence structure found between the covariates in the VAT data set. One common restriction when generating such dependent data is that the marginal distributions must all be of the same family. By using copulas to model dependence between covariates we effectively remove this restriction on the marginal distributions. Additionally, such a construction allows for a wider range of possible dependence structures among the covariates to be studied. We use this data generation procedure to study the performance of regularization methods in cases when attributes are highly dependent in Chapter 6. Using much of the same framework, we study how uneven class distributions in addition to highly dependent covariates affects both predictive performance and variable selection for logistic regression in Chapter 7. In addition, we study the effects of applying three re-sampling methods introduced in Chapter 4 to see if these can improve either prediction or variable selection.

In the final chapter we look closer at the VAT data set. We apply the insight gained from the simulations studies in Chapter 6 and Chapter 7 to model VAT anomaly detection on a real data set. We will also discuss both selection of attributes and the predictive performance of the final model. Discussion and conclusions can be found in Chapter 9.

Chapter 2

Statistical framework

2.1 Mathematical representation of data

Before we begin defining our statistical model, we specify how one can represent the problem of VAT anomaly detection in a mathematical framework. The problem at hand is essentially that of recognizing a given VAT control as either **anomalous** or **not anomalous**. Other than this, there is no further gradation of the controls. This amounts to a binary situation, and we are thus faced with a binary classification problem. Let Y_i represent whether case number i is anomalous or not. We may then give Y_i the binary representation

$$Y_i = \begin{cases} 1 & \text{if case } i \text{ was anomalous} \\ 0 & \text{if case } i \text{ was not anomalous.} \end{cases}$$

It was mentioned in the introduction that an anomaly may occur as the result of an honest mistake, or fraud. However, for simplicity we will in the remaining part of this thesis only refer to $Y_i = 1$ as an indicator of **fraud** rather than the more general term anomaly. We also introduce a mathematical representation for the attributes belonging to each case. For case number i, let attribute number 1 be given by $\mathrm{x}_{i,1}$, and attribute number 2 by $\mathrm{x}_{i,2}$, and so on. We introduce the vector $\mathbf{x}_i = (\mathrm{x}_{i,1}, \mathrm{x}_{i,2}, \ldots, \mathrm{x}_{i,p})^T$ which contains all p attributes for case number i. Further, when we have n cases in our data set we can set up a matrix notation for our data, given by

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}, \qquad \mathbf{X} = \begin{pmatrix} \mathrm{x}_{1,1} & \mathrm{x}_{1,2} & \cdots & \mathrm{x}_{1,p} \\ \mathrm{x}_{2,1} & \mathrm{x}_{2,2} & \cdots & \mathrm{x}_{2,p} \\ \vdots & \ddots & \vdots & \\ \mathrm{x}_{n,1} & \mathrm{x}_{n,2} & \cdots & \mathrm{x}_{n,p} \end{pmatrix} = \begin{pmatrix} \mathbf{x_1}^T \\ \mathbf{x_2}^T \\ \vdots \\ \mathbf{x_n}^T \end{pmatrix}.$$

However, in the following we will often add an additional 1 as the first element of $\mathbf{x}_i$ for all $i = 1, \ldots, n$, and thus the matrix $\mathbf{X}$ becomes an $n \times (p+1)$ matrix. Thus, each row represents the available information we have on a specific case, and each column represents the different covariates. We have now established the mathematical representation of our data set, and can move on to describing how one can model such binary classification problems.

2.2 Modeling probabilities

We assume that the outcome Y_i of each case is binary, and that the probability of fraud in a case given the attributes/covariates is $P_i = P(Y_i = 1|\mathbf{x}_i)$ for case i. The question is now what parametric assumptions to make on the probability P_i. The most common regression model is perhaps linear regression, which has been applied successfully in many different fields. However, modeling probabilities with a linear regression framework is problematic. Assuming a linear regression model for P_i we have $P_i = \eta(\mathbf{x}_i)$, where $\eta(\mathbf{x}_i)$ is the linear predictor defined as

$$\eta(\mathbf{x}_i) = \beta_0 + \beta_1 \mathrm{x}_{1,i} + \cdots + \beta_p \mathrm{x}_{p,i} = \mathbf{x}_i^T \boldsymbol{\beta},$$

with $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$. A problem with this model is that $\eta(\mathbf{x}_i)$ is defined not only on $(0, 1)$, but on the whole of $\mathbb{R}$. This means that our model may predict probabilities outside the range $(0, 1)$ making the interpretation of these predictions problematic. One may then proceed by considering transformations of $\eta(\mathbf{x}_i)$ instead. Such transformations are called (inverse) link functions in the Generalized Linear Model (GLM) framework. There are several link functions one may consider. One option is the cumulative distribution function $\Phi(\cdot)$ of the standard normal distribution, which gives us the model

$$P(Y_i = 1|\mathbf{x_i}) = \Phi(\eta(\mathbf{x_i})),$$

which is commonly called probit regression. Another option is to apply the complementary log-log link from the GLM framework (Jong and Heller, 2008), such that

$$P(Y_i = 1|\mathbf{x_i}) = 1 - \exp(-\exp(\eta(\mathbf{x_i}))).$$

Alternatively, one may use the cumulative distribution function of a logistic distribution with mean 0 and variance $\pi^2/3$ (Balakrishnan, 1991),

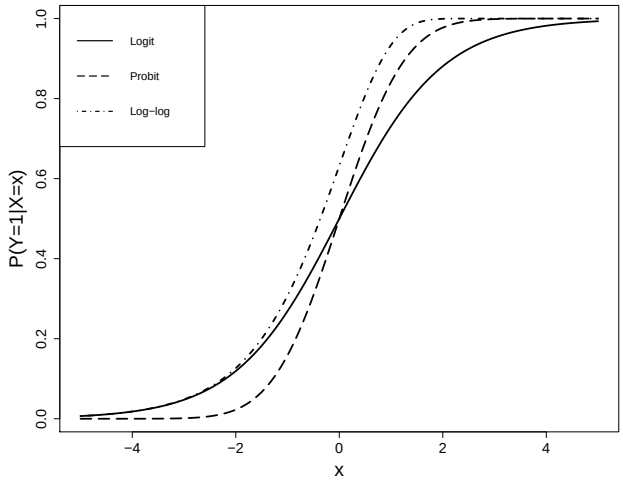


Figure 2.1: $P(Y_i = 1|\mathbf{x}_i)$ plotted using three different transformations of $\eta(\mathbf{x}_i)$.

corresponding to the logit link

$$P(Y_i = 1|\mathbf{x_i}) = \frac{e^{\eta(\mathbf{x_i})}}{1 + e^{\eta(\mathbf{x_i})}}, \tag{2.1}$$

which results in logistic regression. All these transformation ensure that $P(Y_i = 1|\mathbf{x_i}) \in (0, 1)$, though the third is the one most commonly used in practice. Figure 2.1 shows $P(Y_i = 1|\mathbf{x}_i)$ as a function of $\mathbf{x}_i$ for different link functions, with $\beta_0 = 0$ and $\beta_1 = 1$. As can be seen from this figure, the logistic regression model assigns more probability further out in the tails of the distribution than does probit regression. We will in this and later chapters assume the logit link function.

An interesting property for any classification method is its decision boundary, which is briefly discussed here to illustrate the properties of logistic regression. The decision boundary is a line (when $p = 2$) in the predictor space where our model considers it equally likely for a given prediction $\hat{Y}_i$ being a 1 or a 0. In some cases, one would decide to predict $\hat{Y}_i = 1$ when the predicted probability of fraud is greater than 0.5, in which case the decision boundary would be defined by

$$\begin{aligned} P(Y_i = 1|\mathbf{x}_i) &= P(Y_i = 0|\mathbf{x}_i) \\ P(Y_i = 1|\mathbf{x}_i) &= 1 - P(Y_i = 1|\mathbf{x}_i). \end{aligned}$$

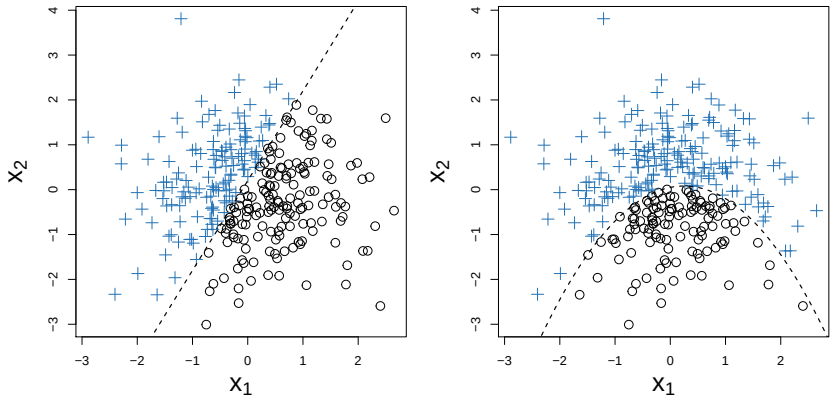


Figure 2.2: Two different decision boundaries for a logistic regression model. On the left: linear decision boundary. On the right: quadratic decision boundary.

By inserting the expression for $P(Y_i = 1|\mathbf{x}_i)$ in the logistic regression model, we find that the decision boundary satisfies

$$\frac{e^{\eta(\mathbf{x}_i)}}{1 + e^{\eta(\mathbf{x}_i)}} = \frac{1}{1 + e^{\eta(\mathbf{x}_i)}} \Rightarrow \eta(\mathbf{x}_i) = 0.$$

Because $\eta(\mathbf{x}_i)$ is a linear function of $\mathbf{x}_i$ the decision boundary is also linear. This may be considered a disadvantage of the logistic regression model if one suspects the decision boundary to be non-linear. However, this can be taken into account by introducing transformations of the individual elements in $\mathbf{x}_i$, while still allowing $\eta(\mathbf{x}_i)$ to be a linear function of $\mathbf{x}_i$, see Figure 2.2. Another issue with logistic regression arises if one wishes to include interaction terms in the model, which requires manual specification of the specific interaction terms to be included. This can cause problems, especially in situations where p already is large, as including all possible two-way interactions leads to a drastic increase in the number of parameters to be estimated.

Assume that the outcome of the cases are independent of each other, and that the outcome of any given case Y_i given its attributes $\mathbf{x}_i$ follows a Bernoulli distribution where $p_i = P(Y_i = 1|X = \mathbf{x}_i)$ is the probability of case i being fraudulent, defined as in (2.1) for $i = 1, \ldots, n$. The likelihood function is then defined as

$$\mathcal{L}(\boldsymbol{\beta}) = \prod_{i=1}^{n} \left(\frac{e^{\eta(\mathbf{x_i})}}{1 + e^{\eta(\mathbf{x_i})}} \right)^{y_i} \left(\frac{1}{1 + e^{\eta(\mathbf{x_i})}} \right)^{1-y_i} = \prod_{i=1}^{n} \frac{e^{\eta(\mathbf{x_i})y_i}}{1 + e^{\eta(\mathbf{x_i})}}. \tag{2.2}$$

Because it is difficult to find parameters that maximize likelihoods such as (2.2), one considers instead the log-likelihood. This has the advantage of being both easier to work with analytically, but also more stable when optimizing numerically. Importantly, the values that maximize a likelihood also maximize the log-likelihood. The log-likelihood function $\ell(\boldsymbol{\beta}) = log(\mathcal{L}(\boldsymbol{\beta}))$ is in this case given by

$$\begin{aligned}\ell(\boldsymbol{\beta}) &= \sum_{i=1}^{n} y_i \eta(\mathbf{x_i}) - log\left(1 + e^{\eta(\mathbf{x}_i)}\right) \\ &= \sum_{i=1}^{n} y_i \mathbf{x}_i^T \boldsymbol{\beta} - log\left(1 + e^{\mathbf{x}_i^T \boldsymbol{\beta}}\right). \end{aligned} \tag{2.3}$$

The maximum likelihood estimator $\hat{\boldsymbol{\beta}}$ is obtained by maximizing (2.3) which must be done using some numerical procedure. A common choice is the Newton-Raphson algorithm. We will in this thesis not concern ourselves with such numerical optimization problems, but use the built-in procedures given in R. Fitting the logistic regression model using maximum likelihood estimation has several advantages. This provides us with a number of already established theoretical results we can use to make inference on $\boldsymbol{\beta}$. One such result of particular importance says that the estimator $\hat{\boldsymbol{\beta}}$ is approximately unbiased and approximately normally distributed,

$$\sqrt{n}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}) \approx N(\mathbf{0}, \mathcal{I}(\beta)^{-1}),$$

where $\mathcal{I}(\boldsymbol{\beta}) = -E\left[\frac{\partial^2}{\partial \boldsymbol{\beta}^2}\ell(\boldsymbol{\beta})\right]$ is known as the Fisher information. This allows for construction of confidence intervals, and performing hypthesis testing for the significance of the coefficients $\beta_1, \ldots, \beta_p$.

Logistic regression suffers when the number of covariates p is large. Including too many covaraites in our model may lead to a model which is highly dependent on the randomness present in our data, rather than the true underlying effects which we are interested in. The problem is twofold: First, one wishes to identify which covariates actually contribute to the distribution of the response Y. Second, one may wish to reduce the effect a covariate has on the response if the corresponding parameter estimate is riddled with variance. These issues are highly relevant within fraud detection, where the number of explanatory variables may be in the hundreds or even thousands, many of which may be highly correlated or have little to no impact on the response. We will in the next chapter give an overview of some of the common techniques and approaches for variable selection and model regularization.

2.3 The bias-variance trade-off

An important concept in most statistical modeling is that of bias-variance trade-off. Parameter selection is part of this problem when constructing our logistic regression model for fraud detection. Including all covariates in the model would yield a model that is too variable w.r.t. the covariates and thus it has low bias but high variance. On the other hand including only a few significant covariates (assuming these could be identified) could lead to a model that is not variable enough, i.e. it has a low variance but high bias. Consider the linear regression setting, where $Y = f(\mathbf{x}) + \epsilon$ for $\epsilon \sim N(0, \sigma^2)$. The objective is to estimate $f = f(\mathbf{x})$ by some function $\hat{f} = \hat{f}(\mathbf{x})$. It may be shown that the expected squared loss can be written as

$$E\left[(Y - \hat{f})^2\right] = \mathrm{Var}(Y) + \mathrm{Var}(\hat{f}) + (f - \mathrm{E}\left[\hat{f}\right])^2. \tag{2.4}$$

Hence, the expected loss may be considered as the sum of three components, namely irreducible error $\mathrm{Var}(Y)$, variance of our prediction $\mathrm{Var}(\hat{f})$ and the squared bias of our prediction $(f - \mathrm{E}\left[\hat{f}\right])^2$. This decomposition essentially illustrates the problem: we wish to minimize (2.4) by reducing the two latter components in the sum. However, minimizing both the variance and bias simultaneously is usually not possible, and one must instead seek to find an optimal relationship between the two. As stated in Hastie et al. (2009), a similar relationship between bias and variance is present when modeling probabilities. A common term in variable selection is that of model complexity. Model complexity is related to the number of parameters in our logistic regression model. A model with a high number of parameters is considered to be more complex than one with only a few. A typical view of how bias and variance are related to model complexity is provided in Figure 2.3. This illustrates that as the model complexity increases, the bias is reduced, but at the cost of increased variance.

2.4 Model selection criteria

The squared loss function was considered for the purpose of illustrating the bias-variance trade-off. For binary classification there are other measures of performance to consider. Deciding which one is most appropriate for a given situation is not always straightforward. First, note that our primary objective is prediction which should be kept in mind when choosing the performance measure. One common way to assess model predictive performance is to consider the accuracy of a model. Assume we have some model $\hat{f}(\mathbf{x}_i)$ that

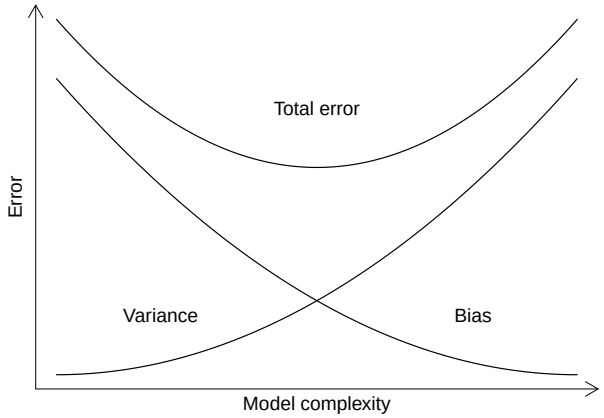


Figure 2.3: An illustration of the tradeoff between bias and variance for the squared loss function.

produces $\hat{P}(Y_i = 1|\mathbf{x}_i)$ for each $\mathbf{x}_i$, so that $\hat{Y}_i = 1$ if $\hat{P}(Y_i = 1|\mathbf{x}_i) \geq c$ for some constant $c \in [0, 1]$, and $\hat{Y}_i = 0$ else. The accuracy is then given by

$$\text{ACC} = \frac{1}{n} \sum_{i=1}^{n} I(\hat{Y}_i = Y_i).$$

This measure is easy to interpret and seems a reasonable one if we are simply interested in the predicted labels (i.e. fraud/not fraud), and not the corresponding probabilities. However this is not the case in fraud detection and accuracy is therefore not appropriate as a measure of predictive performance. This measure does not take into account how certain we are that a given claim is fraudulent. Whether the probability of a claim being fraudulent is estimated to 0.5 or 0.99 is not relevant when using accuracy with $c = 0.5$ as a measure of predictive performance, because we will classify $\hat{Y}_i = 1$ in both cases. Additionally we do not want to classify each case as either fraudulent or not, we wish to provide a probability of a case being fraudulent.

An alternative is the Brier score (Brier, 1950), which does take into account how certain we are that a given claim is fraudulent. The Brier score may be defined as

$$\text{BS} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{P}(Y_i = 1|\mathbf{x}_i) \right)^2.$$

Each term of the Brier score is maximized when $Y_i = 1$ and $\hat{P}(Y_i = 1|\mathbf{x}_i) = 0$, or $Y_i = 0$ and $\hat{P}(Y_i = 1|\mathbf{x}_i) = 1$. In either case the corresponding term of

the Brier score will be 1. Since each term may have a maximum value of 1 and there are n of these, the Brier score like the ACC takes on values between 0 and 1, where a lower value of Brier score indicates better predictive performance.

One point of concern regarding both accuracy and Brier score as measures of predictive performance is their dependence on the ratio of number of fraudulent cases to the number of non-fraudulent cases. Using these measures as criteria for model selection in classification problems where there is an unequal number of fraudulent and non-fraudulent observations in the data set can result in sub-optimal models. For instance, in the VAT data set the ratio of fraudulent to non-fraudulent cases is roughly 1 to 5. A model which classifies $\hat{Y}_i = 0$ for all observations thus obtains an accuracy of 80%. However, such a model has no value if one is interested in detecting fraud. It is therefore easy to be fooled by measures such as accuracy when there is an unequal number of observations associated with the two classes. This can be problematic if one is interested in keeping track of the performance of a model over time. If there is substantial variation in the number of fraudulent cases compared to non-fraudulent cases in different time periods, then a measure such as accuracy will reflect this. One then risks drawing the possibly erroneous conclusion that the performance of a model has changed, simply due to a change in the frequency of anomalous cases. In fraud detection generally one needs to update any predictive model in order to adapt to the changing strategy of fraudsters, as pointed out by Bolton and Hand (2002). It is therefore crucial to evaluate the predictive performance of a model based on a measure that is independent of the number of fraudulent to non-fraudulent observations in order to get a clear picture of the true predictive performance of a model.

As an alternative, one could study how a model performs on predicting the fraudulent and the non-fraudulent cases separately. Hence we may wish to distinguish between which cases are correctly predicted as fraudulent, known as true positives, and which cases are correctly predicted as not fraudulent, known as true negatives. A model with good predictive abilities should achieve a high rate of both true positives and true negatives. Such statistics may be represented by constructing tables such as Table 2.1, commonly referred to as a confusion matrix. Observations correctly classified are found on the diagonal, and the observations incorrectly classified on the off-diagonal. The advantage of such a confusion matrix is that it gives us a view of what types of errors our model makes. We can thus keep track of how well our model is recognizing the fraudulent cases as well as the

		Predicted	
		Fraud	Not fraud
Actual	Fraud	True positives	False negatives
	Not fraud	False positives	True negatives

Table 2.1: Layout of a confusion matrix

non-fraudulent cases, instead of how well our model is at predicting all cases in general. In the previous example where we constructed the model $\hat{Y}_i = 0$ $\forall i$, the number of true negatives would be high, whereas the number of true positives would be 0. Construction of such tables therefore gives a more transparent view of the performance of our model. By adjusting the threshold of predicting a claim as fraudulent, we can create several tables such as Table 2.1 for different thresholds, however it is more common to create a Reciever Operating Characteristic (ROC) curve (Swets, 1988). ROC curves are much used as measures of predictive performance in classification. A ROC curve visualizes how the true positive rate changes as the false positive rate is increased. Partly what makes ROC attractive is that it is based solely on the true positive rate and the false positive rate from Table 2.1, and is thus independent of the underlying class frequencies in the data set (Swets, 1988). This property of the ROC makes it particularly relevant as a measure of predictive performance in fraud detection, since the frequency of fraudulent observations tends to be much lower than that of non-fraudulent observations (Bolton and Hand, 2002). It is important to be aware of the possible shortcomings of ROC curves as measures of performance. One of these is that a ROC curve does not indicate how well the model fits the data. Additionally, if one is interested in training a model which produces correct probabilities for a given event, using the ROC as a measure of performance is inappropriate. However, as pointed out in Chapter 1 we are not interested in the probabilities per se, but rather the probabilities compared against one another. Our objective is to create an ordering of which cases are more likely to be fraudulent, whose true purpose is to separate the fraudulent cases from the non-fraudulent ones. For such objectives, ROC curves are ideal as measures of performance (Fawcett, 2006)

Although a visual display of the ROC by plotting ROC curves may be used to evaluate the predictive performance of a model, it is often desirable to have quantitative means of evaluating the ROC. The typical approach is to calculate the Area Under the (ROC) Curve (AUC). A model which is able to perfectly predict all observations will have an AUC of 1, while randomly

assigning labels as either 0 or 1 with equal probabilities will yield an AUC of 0.5 in a binary classification problem. The AUC has a probabilistic interpretation which helps understand when it is appropriate to use such a measure. It equals the probability that a fraudulent case will be assigned a higher probability of being fraudulent than a non-fraudulent case (Fawcett, 2006).

Regardless of which of the above model selection criteria one uses, one must be careful when using the same data set to both fit and evaluate the predictive performance of a model. The reason for this being that the model might become tailored to that particular data set. This is known as overfitting. Overfitting occurs when a model has to a large degree been fitted to the randomness present in the data. A typical cause of this is the inclusion of too many parameters in a parametric model such as the logistic regression model.

2.5 Training and test set

In an attempt to prevent over-fitting, one can split the original data set into two disjoint subsets: a training set and a test set. The model is estimated/trained on the training set and its predictive performance evaluated on the test set. Because the test set is independent of the training set, one thus obtains an appropriate measure of model performance. How large the training and test sets should be must be evaluated for each individual setting. If the test set is not sufficiently large, it might not be representative of the underlying data thus providing poor basis for evaluation of model error. However, choosing a large test set comes at the cost of a smaller training set. This is unfortunate since the test set is not used at all to fit the model, and we therefore in a sense lose data.

2.6 K-fold cross-validation

While for some methods it may be adequate to train a model on the training set and evaluate the model on the test set, this is not always the case. For some of the methods which we will discuss later we may wish to estimate the predictive performance on the test set in order to tune our model. Such an estimate can be obtained by dividing the original training set into K approximately equally sized subsets $\{\mathbf{Y}_1, \mathbf{X}_1\}, \ldots, \{\mathbf{Y}_K, \mathbf{X}_K\}$ often referred to as folds. One would then use $\{\mathbf{Y}_2, \mathbf{X}_2\}, \ldots, \{\mathbf{Y}_K, \mathbf{X}_K\}$ as the training set

and $\{\mathbf{Y}_1, \mathbf{X}_1\}$ as the test set, independent of the training set. The next step is to fit a model using $\{\mathbf{Y}_1, \mathbf{X}_1\}, \{\mathbf{Y}_3, \mathbf{X}_3\}, \ldots, \{\mathbf{Y}_K, \mathbf{X}_K\}$ as the training set, and $\{\mathbf{Y}_2, \mathbf{X}_2\}$ as the test set. This procedure is repeated for all the K subsets, thus obtaining K estimates of model performance. Seeing that our final model will be used in a predictive manner, this approach seems a reasonable one since it to a certain degree mimics the situation in which the final model will be used and evaluated. The independence between the test and training set at each step of this method is important because it results in unbiased estimates of model performance for each of the K models. The K estimates are then averaged to get a final estimate of model performance. We can use the Brier score to evaluate the model fit on each subset. The average of these K Brier scores will be denoted

$$\mathrm{BS}_{\mathrm{CV}(K)} = \frac{1}{K} \sum_{k=1}^{K} \mathrm{BS}(k),$$

where $\mathrm{BS}(k)$ is the Brier score based on the k'th subset. An alternative is the cross-validated AUC given by

$$\mathrm{AUC}_{\mathrm{CV}(K)} = \frac{1}{K} \sum_{k=1}^{K} \mathrm{AUC}(k).$$

There is great flexibility in the choice of K. As in variable selection, it is a matter of balancing the bias-variance trade-off. Choosing $K = 2$ leads to a situation close to that of dividing the data into just one training set and one test set, resulting in high bias but low variance. On the other hand, setting $K = n$ (leave-one-out cross-validation) results in low bias but high variance. Besides this, the computational demands increase with K, since models must be fitted and their predictive performance estimated K times. The effect K has on estimating model error was extensively studied in Kohavi (1995), where it was concluded that $K = 10$ provides an optimal trade-off between variance and bias. Further, it was recommended to perform stratified K-fold cross validation as opposed to regular K-fold cross validation to reduce bias in the estimate of model error. This means that the subsets $\{\mathbf{Y}_1, \mathbf{X}_1\}, \ldots, \{\mathbf{Y}_K, \mathbf{X}_K\}$ should be constructed such that all subsets are approximately equal w.r.t. the number of fraudulent and non-fraudulent cases in each fold, rather than performing a completely random split of the data.

Algorithm 2.1 Stratified cross validation

Input: Y, K

1: $\mathcal{I}_0 = \text{which}\,(Y == 0)$
2: $\mathcal{I}_1 = \text{which}\,(Y == 1)$
3: $n_0 = \text{length}\,(\mathcal{I}_0)$
4: $n_1 = \text{length}\,(\mathcal{I}_1)$
5: $\text{foldSize}_0 = n_0/K$
6: $\text{foldSize}_1 = n_1/K$
7: $\text{labels} = \text{rep}(0, n_0 + n_1)$
8: $\text{labels}_0 = \text{c}\,(\text{rep}(1, \text{foldSize}_0)\% * \%t(1:K))$
9: $\text{labels}_1 = \text{c}(\text{rep}(1, \text{foldSize}_1)\% * \%t(1:K))$
10: $\text{labels}\,[\mathcal{I}_0] = \text{sample}(\text{labels}_0)$
11: $\text{labels}\,[\mathcal{I}_1] = \text{sample}(\text{labels}_1)$
12: **Return** labels

Algorithm 2.1 performs stratified cross validation. Note especially how the sampling of positive and negative observations is done separately to obtain an even number of both classes in all folds. This algorithm assumes that the positive and negative observations can be divided evenly across all folds, an assumption that needs not be taken into considerations with just small modifications to the algorithm.

Chapter 3

Model regularization

Model regularization for regression problems is a wide subject which we can not cover in its entirety in this thesis. However, we will present some of the more common methods. Generally, regularization methods are applied in order to reduce overfitting. This is done by reducing the number of parameters, or shrinking the parameters in a model. Methods that are intuitively quite clear and can in some cases perform reasonably well are forward and backward selection. These methods can however often result in sub-optimal models. Additionally, it has been observed that models trained by such an approach tend to be highly variable (Breiman, 1996b). This is because small changes in the training data set can cause substantial changes to the final model. Alternatively, one can transform the data prior to model training to reduce the dimension of the problem. For instance the $n \times p$ matrix $\mathbf{X}$ can be transformed using a vector $\theta = (\theta_1, \ldots, \theta_p)^T$ to produce the $n \times p$ matrix $\mathbf{Z} = \theta\mathbf{X}$. One then proceeds by modeling the response Y using only $p^* < p$ of the columns in $\mathbf{Z}$. One common such method is Principal component regression, Hastie et al. (2009).

The focus of this chapter will instead be methods that seek to optimize (2.3) in a constrained manner. Many such methods have been proposed, where the main difference between these lie in how one formulates the imposed constraints.

3.1 Ridge regression

One of the first methods introduced for model regularization is ridge regression. The idea behind ridge regression is still to optimize our log-likelihood w.r.t. $\boldsymbol{\beta}$, but to do so under a variance reducing constraint. Let the log-likelihood be defined as in (2.3), however instead of optimizing (2.3) one

now seeks to optimize (Le Cessie and Van Houwelingen, 1992)

$$\sum_{i=1}^{n} y_i \eta(\mathbf{x}_i) - log\left(1 + e^{\eta(\mathbf{x}_i)}\right) - \lambda \sum_{j=1}^{p} \beta_j^2, \tag{3.1}$$

where the last term is a penalization term whose effect depends on λ, known as a penalty parameter. If $\lambda = 0$, (3.1) becomes (2.3), and we are simply left with maximum likelihood estimation. As λ increases, our model is increasingly forced to reduce the sum $\sum_{j=1}^{p} \beta_j^2$ in an optimal way. We thereby introduce some bias to the estimates $\hat{\beta}_1, \ldots, \hat{\beta}_p$ in exchange for lower variance and hopefully better predictive performance. We are in essence exploring the possible gains in predictive performance by trading variance for bias as depicted in Figure 2.3. When adding penalization terms such as in (3.1), it is common to standardize all covariates so they have a mean of zero, and a standard deviation of one. This ensures that the penalty introduced by λ shrinks all coefficients equally, independent of the scale of their respective covariates.

For the purposes of presenting some theoretical results, consider now a linear regression model where one assumes

$$\mathbf{Y} = \beta_0 + \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where $\mathbf{X}$ is an $n \times p$ matrix, so the first column containing $1's$ has been removed. Further, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T$ and $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^T$ is a vector containing the residuals which are all assumed independent and identically distributed according to $N(0, \sigma^2)$. In matrix notation, the objective function to be minimized is now

$$(\mathbf{Y} - \beta_0 - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{Y} - \beta_0 - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}.$$

One may then show that the ridge estimator is

$$\begin{aligned} \hat{\beta}_0 &= \frac{1}{n} \sum_{i=1}^{n} y_i \\ \hat{\boldsymbol{\beta}} &= (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}. \end{aligned}$$

We can see that the ridge estimator $\hat{\boldsymbol{\beta}}$ differs from the regular least squares estimator, as it includes an additional term that depends on λ. From this result we see that increasing the value for λ shrinks $\hat{\boldsymbol{\beta}}$. While this result is only valid for linear regression, the same behavior is present for the ridge estimator in logistic regression (Le Cessie and Van Houwelingen, 1992).

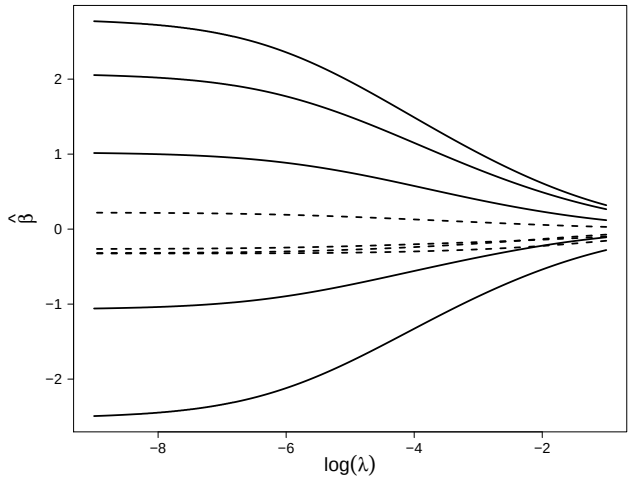


Figure 3.1: Plot of $\hat{\boldsymbol{\beta}}$ coefficients resulting from fitting a logistic regression model with a ridge penalty. True zero-coefficients shown in dashed lines, non-zero coefficients in solid lines.

We consider an example to illustrate the effect λ has on the estimated model coefficients. To do this, we will draw simulations of $\mathbf{x}_i$ for $i = 1, \ldots, n$ with $n = 100$. We will in this example draw 9 explanatory variables so that $\mathbf{x}_i = (1, x_{i,1}, \ldots, x_{i,9})^T$. All explanatory variables were drawn independently from the standard normal distribution. The model was defined as

$$\boldsymbol{\beta} = (3, -2.5, 2, -1.5, 1, 0, 0, 0, 0)^T,$$

with the intercept $\beta_0 = 0$. The next step is to compute $\eta(\mathbf{x}_i) = \mathbf{x}_i^T\boldsymbol{\beta}$ and $P(Y_i = 1|\mathbf{x}_i) = e^{\eta(\mathbf{x}_i)}/\left(1 + e^{\eta(\mathbf{x}_i)}\right)$, and draw $Y_i \sim \text{Bernoulli}(P(Y_i = 1|\mathbf{x}_i))$. We now have data which we can use to estimate a logistic regression model with a ridge penalization term. This was done in R using the package `glmnet`. The model was fit with 100 different values for λ ranging from $\exp\{-9\}$ to $\exp\{-1\}$. In Figure 3.1, we can see the behavior previously discussed, where the estimated model coefficients shrink as λ increases. Note especially how the estimated coefficients shrink; they approach 0 very slowly and will in fact never be estimated to exactly 0. Hence, ridge is not able to perform variable selection. For problems where the true model consists of many variables with small coefficient values ridge regression may therefore be appropriate. On the other hand ridge regression may struggle and result in poor predictive performance when the sizes of the $\boldsymbol{\beta}$ coefficients are more varied, i.e. both small and large coefficients in $\boldsymbol{\beta}$ (Breiman, 1996b). A question that arises when such small parameter estimates are obtained is

whether these parameters truly are close to, but not equal to zero, or if they truly are zero. The effect parameter estimates close to zero have on the predicted values may not be that big, but the interpretation of the model still remains questionable.

3.2 Lasso regression

Lasso regression is another widely used method for model regularization. Again, the idea is to put restrictions on the model coefficients $\beta_1, \ldots, \beta_p$ when performing model estimation. Let the log-likelihood be defined as in (2.3). One now considers (Tibshirani, 1996)

$$\sum_{i=1}^{n} y_i \eta(\mathbf{x}_i) - log\left(1 + e^{\eta(\mathbf{x}_i)}\right) - \lambda \sum_{j=1}^{p} |\beta_j|, \tag{3.2}$$

where again if $\lambda = 0$, (3.2) becomes (2.3), which leads to regular maximum likelihood estimation. Lasso regression differs from ridge regression by replacing the l^2 penalization term $\sum_j \beta_j^2$ with an l^1 penalization term $\sum_j |\beta_j|$. This leads to a harder penalty for small values of β_j, $j = 1, \ldots, p$. Using lasso regression, one may therefore end up with coefficient estimates which are exactly zero. Lasso as opposed to ridge thus performs variable selection. More precisely, because the lasso penalty unlike the ridge penalty is singular at 0 (its derivative is not defined), the lasso can perform variable selection (Fan and Li, 2001).

Let us also here consider an example using simulated data, with the procedure for generating data the same as before. Figure 3.2 shows the result of fitting a logistic regression model with lasso penalty. As λ increases the coefficients shrink, with some estimates even becoming exactly zero. Thus, lasso regression also performs variable selection by excluding some predictors from our model, simplifying its interpretation. This is considered a particularly attractive property of lasso. However, Figure 3.2 also illustrates that while lasso does shrink the zero-coefficients to zero, it introduces potentially large bias to the remaining non-zero coefficients. By the time all of the zero-coefficients have been correctly estimated as zero, the remaining non-zero coefficients have been shrunk to only a fraction of their original size. In fact, for a large coefficient β_j it has been shown that lasso will produce biased estimates. This is owed to the fact that the derivative of $|\beta_j|$ does not equal zero when β_j is large (Fan and Li, 2001), which is considered a weakness of lasso. Further, in situations with grouped variables, i.e. when groups of covariates are highly correlated lasso is likely to select

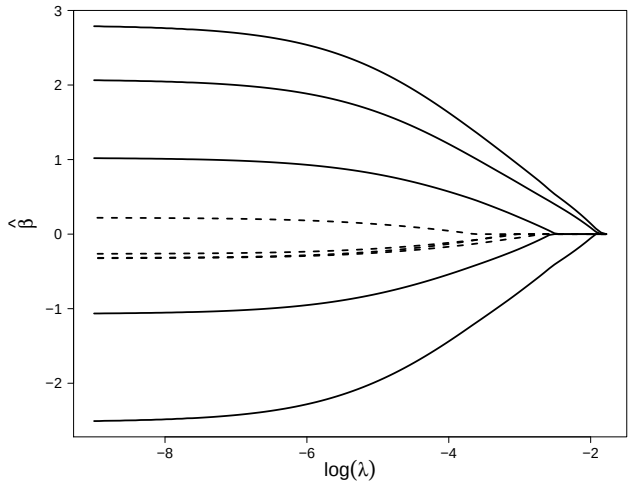


Figure 3.2: Coefficient paths for varying values of λ using the lasso penalty. Dashed lines for zero-coefficients, solid lines for the non-zero coefficients.

only one of these with little regards to which one it is (Zou and Hastie, 2005). Additionally, it has been observed that lasso may perform worse in terms of prediction than for instance ridge when covariates are highly correlated (Tibshirani, 1996). Alternative methods for regularization have therefore been proposed. One such menthod which addresses particularly the last two points is elastic net.

3.3 Elastic net

Proposed by Zou and Hastie (2005), the elastic net combines the penalties used in lasso and ridge, giving the objective function

$$\sum_{i=1}^{n} y_i \eta(\mathbf{x}_i) - log\left(1 + e^{\eta(\mathbf{x}_i)}\right) - \lambda \left(\alpha \sum_{j=1}^{p} |\beta_j| + (1-\alpha) \sum_{j=1}^{p} \beta_j^2 \right). \qquad (3.3)$$

This combination of ridge and lasso introduces an additional parameter α to be optimized. This parameter controls how much emphasis should be put on either the ridge or lasso penalization terms. Nonetheless, as long as $\alpha > 0$ elastic net like lasso performs variable selection due to the singularity at 0 inherited from the lasso penalty. The elastic net was proposed as an improvement on lasso, with the motivation being that the lasso may struggle in terms of both predictive performance and variable selection when variables are highly correlated. The elastic net does not suffer from

this to the same degree, which has been illustrated both theoretically and empirically in Zou and Hastie (2005). Simulation studies conducted in the same paper illustrated that the elastic net performs better than lasso in terms of prediction when collinearity is present.

Note that we will in this thesis focus on the so-called *naive* elastic net in the original paper by Zou and Hastie (2005). Other works have focused exclusively on this version of elastic net (Friedman et al., 2010), so we will do the same in this thesis. Additionally, in the `glmnet` package in R which has Trevor Hastie as its maintainer, only the naive elastic net has been implemented.

Another alternative to lasso, the adaptive lasso, has been proposed to address the problem of biased estimation of large $\boldsymbol{\beta}$ coefficients.

3.4 Adaptive lasso

Being largely similar to the lasso the adaptive lasso proposes an individual penalty for each coefficient β_j, resulting in the objective function (Zou, 2006)

$$\sum_{i=1}^{n} y_i \eta(\mathbf{x}_i) - log\left(1 + e^{\eta(\mathbf{x}_i)}\right) - \lambda \sum_{j=1}^{p} \mathrm{w}_j |\beta_j|. \tag{3.4}$$

The question is then how the weights w_j should be chosen. It is shown in Zou (2006) that if the weights w_j are determined from the data then some of the theoretical shortcomings of lasso can be fixed. These weights may be estimated by $\hat{\mathrm{w}}_j = 1/|\hat{\beta}_j^*|^\gamma$, with $\gamma > 0$ and where $\hat{\beta}_j^*$ for $j = 1, \ldots, p$ are the maximum likelihood estimators. Alternatively one can compute the weights using the ridge estimates. The latter is recommended particularly when collinearity is present in the data because of the greater stability of ridge over ML estimators in such situations. Thus the penalty applied to coefficient j is adjusted based on the initial estimate $\hat{\beta}_j^*$, where a lower initial estimate results in a greater penalty for the respective coefficient in (3.4). Heuristically, the advantage of adaptive lasso over lasso is because the initial estimates $\hat{\beta}_j^*$ used to compute the weights will tend to 0 for the coefficients that are truly zero, as n increases. The inclusion of the weights w_j gives adaptive lasso the so-called oracle properties, a set of properties which we will later discuss.

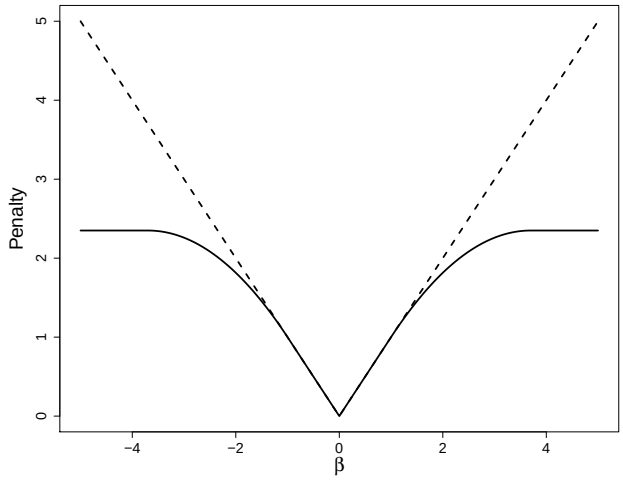


Figure 3.3: Penalty as a function of β_j for SCAD and lasso. $a = 3.7$ and $\lambda = 1$ in both cases. Solid line is the SCAD penalty; dashed line is the lasso penalty.

3.5 SCAD regression

Lastly, the Smoothly Clipped Absolute Deviation (or SCAD in short) penalty may be used, resulting in the objective function (Fan and Li, 2001)

$$\sum_{i=1}^{n} y_i \eta(\mathbf{x}_i) - log\left(1 + e^{\eta(\mathbf{x}_i)}\right) - \sum_{j=1}^{p} p_\lambda(\beta_j; a), \tag{3.5}$$

where for $a > 2$ and $\lambda > 0$

$$p_\lambda(\beta_j; a) = \begin{cases} \lambda|\beta_j| & \text{if } |\beta_j| \leq \lambda \\ \frac{2a\lambda|\beta_j| - \beta_j^2 - \lambda^2}{2(a-1)} & \text{if } \lambda < |\beta_j| \leq a\lambda \\ (a+1)\lambda^2/2 & \text{if } |\beta_j| > a\lambda. \end{cases} \tag{3.6}$$

Hence the penalization term applied to coefficient j depends on the size of β_j. To illustrate the difference between the penalization terms used in lasso and SCAD regression, the penalties from (3.2) and (3.6) have been plotted together for values of β ranging from -5 to 5. From Figure 3.3 we can see that SCAD penalizes similarly to lasso up to a certain point, where it slowly flattens out, eventually becoming constant. So SCAD differs from lasso in that it gives a smaller penalty for greater values of $\hat{\beta}_j$ for $j = 1, \ldots, p$.

Again, let us look at an example using simulated data. The procedure is the same as before, but with the penalization term now as in (3.6). The

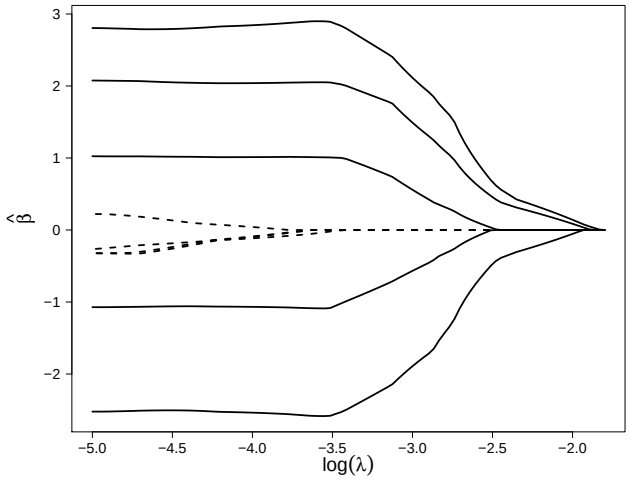


Figure 3.4: Trace plot of the coefficient values $\hat{\beta}_j$ as a function of λ.

package `ncvreg` was used to fit the model using the SCAD penalty. Figure 3.4 shows that the behavior of the estimated coefficients $\hat{\beta}_j$ differs from that in Figures 3.1 and 3.2. For this specific situation at least, it appears that the SCAD penalty is more reluctant at shrinking the large non-zero coefficient up to a certain point while simultaneously shrinking the zero-coefficients to zero. Further insight into the behavior of SCAD may be found by differentiating (3.6) w.r.t. β_j, to obtain the rate at which the penalization changes with β_j;

$$p'_\lambda(\beta_j; a) = \begin{cases} \frac{\beta_j}{|\beta_j|}\lambda & \text{if } |\beta_j| \leq \lambda \\ \frac{\beta_j}{|\beta_j|}\left(\frac{a\lambda - |\beta_j|}{(a-1)}\right) & \text{if } \lambda < |\beta_j| \leq a\lambda \text{ for } j = 1, \ldots, p \\ 0 & \text{if } |\beta_j| > a\lambda. \end{cases} \tag{3.7}$$

Hence, the penalty increases by a constant factor when $|\beta_j| \leq \lambda$, and then becomes an increasing function of β_j when $\lambda < |\beta_j| \leq a\lambda$. When $|\beta_j| > a\lambda$ the penalty does not increase, and hence remains constant. We see that for coefficients that satisfy $|\hat{\beta}_j| > a\lambda$ the penalty does not change as $\hat{\beta}_j$ decreases, so shrinking $\hat{\beta}_j$ to a value that still satisfies $|\hat{\beta}_j| > a\lambda$ has no impact on the penalization term. This ensures unbiased estimation of large coefficients (Fan and Li, 2001), a property which we mentioned earlier is lacking for lasso. Further, in the case when $\lambda < |\hat{\beta}_j| \leq a\lambda$, and $\hat{\beta}_j > 0$ the penalty changes by

$$\frac{a\lambda - |\hat{\beta}_j|}{(a-1)} < \frac{a\lambda - \lambda}{(a-1)} = \lambda.$$

Thus, if the penalization term is to be reduced in the most optimal way it would be most beneficial to shrink the parameters that satisfy $\hat{\beta}_j \leq \lambda$ since the rate at which the penalty for these parameters changes is the greatest. Like adaptive lasso, SCAD possesses the oracle properties which we will now present briefly.

3.6 Oracle properties

The oracle properties of an estimator were mentioned in both SCAD and adaptive lasso. We will in this section look at what this means. The explanation of the term oracle property varies somewhat in the literature, but the essence of it is that an estimator possessing the oracle property will correctly identify the non-zero coefficients in $\boldsymbol{\beta}$ with probability tending to 1. In addition, the estimator of the non-zero coefficients is asymptotically normally distributed and unbiased. In mathematical terms, give the $\boldsymbol{\beta}$ vector the representation $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2)$, where $\boldsymbol{\beta}_1 = (\beta_1, \ldots, \beta_{p_n})$ contains all non-zero coefficients, and $\boldsymbol{\beta}_2 = (0, \ldots, 0)$ is a vector of zeroes. Then, one property that an oracle estimator has is

$$P(\hat{\boldsymbol{\beta}}_2 = \mathbf{0}) = 1 \text{ as } n \to \infty.$$

In other words, this means that the zero-coefficients in $\boldsymbol{\beta}_2$ can be identified simultaneously with a probability that tends to 1 as the number of observations $n \to \infty$. The second property that an oracle estimator must possess is that

$$\sqrt{n}(\hat{\boldsymbol{\beta}}_1 - \boldsymbol{\beta}_1) \overset{d}{\approx} N(\mathbf{0}, \boldsymbol{\Sigma}) \text{ as } n \to \infty,$$

for some covariance matrix $\boldsymbol{\Sigma}$. For the necessary assumptions and technicalities regarding the oracle property, as well as the proofs that the SCAD and adaptive lasso indeed possess this property, i refer to Fan and Li (2001) and Zou (2006). These results combined means that an oracle estimator is able to recognize which β_j's are non-zero, and provide unbiased estimates for these β_j's as $n \to \infty$. Applied to our current objective of fraud detection, this means that as $n \to \infty$ the SCAD and adaptive lasso estimators are able to identify which of the predictors truly have effect in modeling of fraud detection, and which predictors that are only correlated to those that do. From an applied point of view, it is particularly interesting to study how well these properties of the SCAD and adaptive lasso estimators hold when the number of observations n does not tend towards infinity, which is something we can not achieve in practice. Leeb and Pötscher (2008) argued that the

presence of oracle properties does not guarantee good performance in the finite sample size setting. They further illustrated their point with several simulation studies in the linear regression setting, in which the SCAD estimator performed worse than the ordinary least squares estimator. Additionally, increasing the sample size did not lead to an increase in performance of the SCAD estimator in comparison to the OLS estimator, but actually worsened. These results do not contradict the oracle properties, but they illustrate that one should not focus too much on the oracle properties when evaluating an estimator. Especially because these are asymptotic properties whose validity it is difficult to ascertain in the finite setting.

3.7 Summing up

We have now discussed some methods for model regularization, where all methods have their strengths and weaknesses. Ridge does not perform variable selection, and thus may be inappropriate when the number of covariates is large. Lasso does perform variable selection by being able to estimate $\hat{\beta}_j$'s as exactly zero. However, lasso may struggle when explanatory variables are highly correlated, since it tends to select too few variables in such situations. Thus, combining the strengths of lasso and ridge, the elastic net was discussed. Finally, we also looked at the adaptive lasso and the SCAD, both of which possess the seemingly attractive oracle property. The first two methods, namely lasso and ridge are often viewed as the classical methods. Both have proven to be useful in many situations. The strength of these models lie, among other things, in their apparent simplicity. They both benefit from the fact that one needs only optimize over one parameter λ. This makes the methods intuitively easy to understand and, perhaps more importantly, makes them quite easy to implement in software such as R. Additionally, since one need only optimize over one parameter, fitting such models can be less computationally expensive.

Chapter 4

Data re-sampling

The second issue that this thesis will discuss is the issue of unbalanced data sets in binary classification problems and how to potentially remedy problems which may arise for such data. A data set is said to be unbalanced if there is a large difference in the number of observations between two classes. The degree of unbalance varies from problem to problem, and there is no single threshold for when a data set is said to be unbalanced. We already discussed potential issues regarding model evaluation in terms of measuring predictive performance in such situations in Section 2.4. However, we did not discuss how an unbalanced training set affects the model training procedure itself, which is the topic of this chapter. In this and and subsequent chapters, we will use the term majority class to denote the class of which there is a majority of observations in the data set. For our application, this means that the non-fraudulent class is our majority class. Similarly, the term minority class is used to denote the class of which there are few observations in the data set, i.e. the fraudulent class.

One example of an unbalanced data set is given in Solberg and Solberg (1996) where one was interested in detecting whether an image depicted an oil spill or only a look-alike of an oil spill. Only 2% of the observations were images of oil spills, and 98% were look-alikes. Several other data sets were presented in Ling and Li (1998), which focuses on the issue of direct marketing. Here the data sets were again highly unbalanced with only 1.2%, 7% and 1% of positive samples in the three data sets given there. Examples of data imbalance in credit card fraud can be found in Chan et al. (1999), who worked with two separate data sets where 15% and 20% of transactions were fraudulent. It was noted in this paper that these figures were most likely artificially high. A review of statistical modeling in many different forms of fraud detection given in Bolton and Hand (2002) states that in the

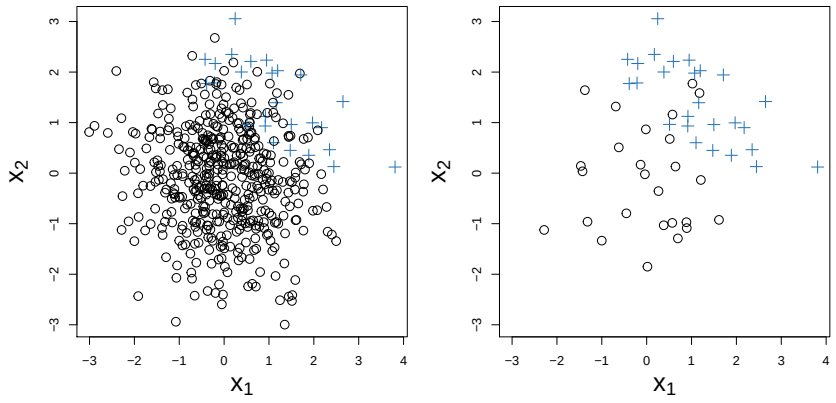


Figure 4.1: Left: Unbalanced data set. Right: balanced data set. Blue + means $Y_i = 1$, black dots means $Y_i = 0$.

domain of money laundering, as little as 0.05% to 0.1% of all transactions may be involved in money laundering. The problem with unbalanced data sets such as those mentioned above is that a model estimated from them can be lacking in terms of predictive ability with respect to the minority class (Kubat and Matwin, 1997).

A data set of one thousand observations was generated to visualize the problem. Two covariates x_1 and x_2 were both sampled from normal distributions with the model $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2) = (-5.5, 2, -2)$. Simulated instances of Y were then drawn as in Chapter 3. With the $\boldsymbol{\beta}$ vector given above, the resulting data set was unbalanced with 5% positive samples. In the right plot of Figure 4.1, only a subset of the negative samples were kept in order to create a situation where the number of negative samples equals the number of positive samples. The plot on the left is thought to reflect a data set one may encounter in situations mentioned in the above examples. Here all negative samples have been included so the data set is highly unbalanced. Remedies for problems stemming from unbalanced data sets such as these may be algorithmic or re-sampling. We will now discuss some of the most commonly used techniques within the latter.

4.1 Under-sampling

A first approach in re-sampling is to under-sample the majority class. This means that we in some way remove a certain number of majority class observations. There are generally two ways of under-sampling a data set. The first, and easiest approach is to remove data points from the majority class with uniform probabilities until some desired majority class size is reached. This is sometimes referred to as random under-sampling. As an alternative to random under-sampling, one can selectively remove majority class observations in order to obtain the desired majority class size. Both random under-sampling and selective under-sampling were tested and compared in Japkowicz (2000). Selective under-sampling was performed by removing only those majority class observations that were far away from the minority class observations. Among the concluding remarks were that random under-sampling could improve the predictive performance of a method, and that selective under-sampling seemed to offer little improvement over random under-sampling. Besides possibly improving the predictive performance of a classification method, under-sampling has one clear advantage in that it reduces the size of the training set to a possibly quite large extent. Say a data set consisting of one million rows with a minority frequency of 1% is under-sampled to contain an equal amount of positive and negative samples; the resulting data set would consist of only twenty thousand observations, thus substantially reducing the computational burden.

Using under-sampling methods either in a random or selective manner may lead to loss of information about the majority class, as was mentioned in Ling and Li (1998). In order to prevent, or at least reduce the loss of information, an alternative re-sampling method is to over-sample the minority class.

4.2 Over-sampling

Over-sampling the minority class may be used either as an alternative to, or together with under-sampling of the majority class. As with under-sampling, this can be done either in a random or selective/synthetic manner. The random over-sampling approach samples uniformly from the observations in the minority class with replacement until a desired minority class size is reached.

Randomly over-sampling the majority class has been criticized for causing

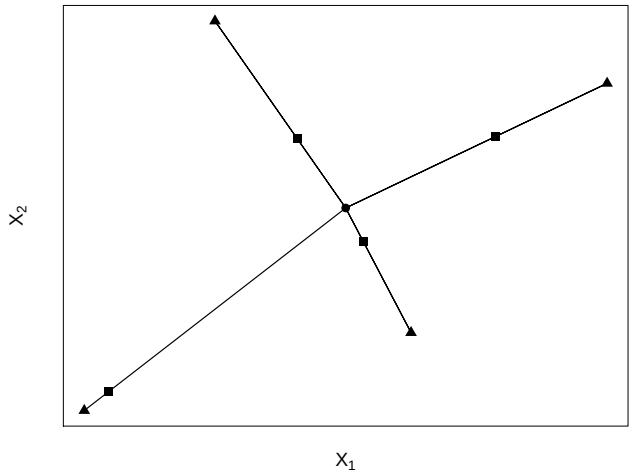


Figure 4.2: An illustration of how synthetic observations are created. Synthetic observations (squares) are added between the minority class observation (dot) and its nearest neighbors (triangles).

overfitting, particularly for tree-based methods (Chawla et al., 2002). A more sophisticated over-sampling method has therefore been proposed in order to prevent over-fitting from occurring and thus increase the predictive performance of a model.

4.2.1 Synthetic minority over-sampling technique

Synthetic minority over-sampling technique (SMOTE) (Chawla et al., 2002) seeks to create new, synthetic minority class observations as opposed to simply sampling with replacement from the ones already in the data set. Over-sampling is done by identifying the k nearest minority class neighbors of a minority class observation which will be chosen to create m new, synthetic observations. One then computes the differences between the minority class observation and its k nearest neighbors. These differences are represented by vectors of length p for a p-dimensional classification problem. Each of these vectors are then multiplied by a uniformly drawn number in $[0, 1]$ and added to the original minority class observation, creating m new synthetic minority class observations. A more illustrative explanation is that one draws lines between the minority class sample in question and the k nearest neighbors, placing m new samples uniformly along these lines. Figure 4.2 illustrates how synthetic minority class observations are created using the approach described above. The dot is the minority class observation,

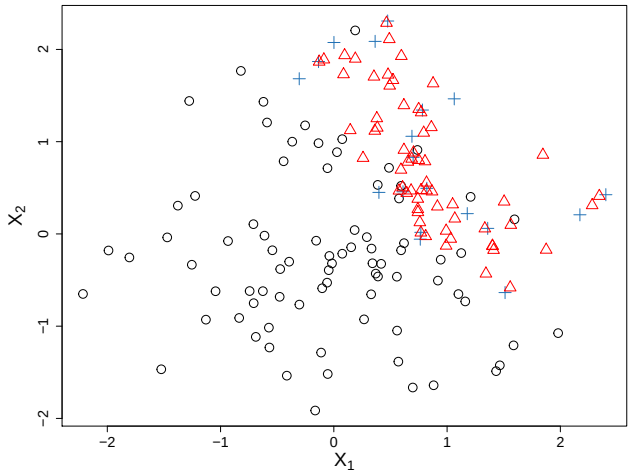


Figure 4.3: Over-sampling using the SMOTE algorithm with k=10. Red triangles are synthetic observations.

with the triangles being the m observations chosen to create new synthetic observations. The vectors of differences are then computed (lines), and one synthetic observation (squares) is generated uniformly along each of these lines. Modifying the training set by introducing new synthetic observations may seem troublesome. However, there are some justifications as to why one would wish to do this. One necessary assumption is that samples close to one another possess approximately the same properties, meaning that samples close to one another in predictor space will be likely to have the same class label. The SMOTE algorithm can thus be thought of a way of filling regions in which the data about the minority class are sparse.

The method SMOTE from the R package `DMwR` can be used to run the SMOTE algorithm. This has been done for the data shown in Figure 4.1 with $k = 10$, and oversampling has been performed until $\bar{Y} = 0.5$. The results are given in Figure 4.3. We can see from Figure 4.3 that the result is a more distinct region of positive class observations. The effect k has on the creation of synthetic samples is shown in Figure 4.4. For $k = 1$, SMOTE creates synthetic samples between a minority sample and only its nearest neighbor, resulting in some possibly undesirable structure in the data (Figure 4.4 right). If $k = 10$, neighbors that are further away are also considered when constructing synthetic samples. What the optimal value of k is in a given situation is not obvious, but using $k = 5$ has been recommended in Chawla et al. (2002).

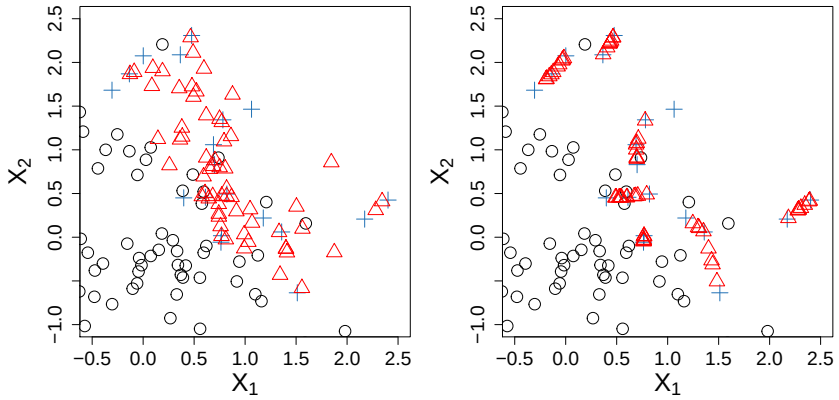


Figure 4.4: A zoomed in view of over-sampling using the SMOTE algorithm with $k = 10$ (left), $k = 1$ (right) and m=2. Red triangles are synthetic observations.

SMOTE was combined with under-sampling in Chawla et al. (2002) and was found to provide better predictive performance in terms of AUC in the vast majority of data sets used. The combination of SMOTE and under-sampling also proved to give better performance than if only under-sampling was used.

Chapter 5

Generating data using copulas

Studying only one data set does not serve as a good basis for learning about the methods discussed in Chapter 3 and Chapter 4. Peculiarities about the data can produce results that are not representative for other situations, and thus do not generalize to other applications. Seeing that our objective is to gain just such a general overview and insight it would be in our interest to study several data sets to see which methods (if any) stand out as best in any given situation. Therefore, the purpose of this chapter is to demonstrate how data sets may be generated in order to carry out such studies in a controlled environment where we know the true model. This makes comparing and contrasting the performance of our model selection and re-sampling methods easier. However, for such an analysis to have any value it is crucial that the generated data set holds to an as large degree as possible the same properties as those found in a real life fraud detection data set. This means that the data sets in our simulations should have a number of properties: a large number of covariates (though still $p < n$), both categorical and numerical covariates, and lastly that the covariates may be highly dependent. Dependence between both numerical and categorical covariates should be taken into consideration. The requirement of dependence between numerical and categorical variables is of some concern. This is because there are are no multivariate probability distributions with both continuous and discrete variables, most distributions are either discrete or continuous. A solution to this problem is to generate the data using a copula approach.

5.1 Preliminary descriptive analysis of tax data

To ensure that our generated data sets are to some degree representative of real fraud detection data sets we will first analyze the VAT data set we introduced in Chapter 1. The focus of this analysis will primarily be on the dependence structure, and the strength of dependence between the covariates. We will then apply our findings from the VAT data set to producing simulated data.

Upon analyzing the covariates in the VAT data set it became apparent that dependence indeed is present to a large degree, and that some covariates seemed to be quite highly dependent. For instance, one set of covariates could all be highly dependent, with some of the covariates in this set being dependent with second set of covariates. The covariates making up this second set of covariates would again be highly dependent with each other. There appeared to be a certain structure in the data with regard to dependence. There may be several reasons why such structures in the covariates can occur. They may well be a result of covariates where dependence naturally occurs. For instance revenue and number of employees can be one such pair of dependent covariates. Dependent covariates may also be the result of feature engineering, in which one tries to construct new covariates in order to extract more information from a data set. Examples of feature engineered variables are the ratio of two existing covariates, which then may be highly correlated with any of the two covariates in that ratio. This motivated a group interpretation of the dependence structure between the covariates. We can then see if such groups of covariates can in fact be found in the VAT data set. In order to do this we must first give some definition of such groups of covariates. We define a group of covariates to be a set of one or more covariates where the dependence between all covariates is greater than some threshold τ in absolute value. In addition, the dependence between all covariates within one group must have the same sign. The definition for what constitutes a group can of course be discussed, and the definition will have an impact in the results obtained. Another important part which will undoubtedly have a large impact on the results is how one chooses to measure dependence. We have in this section chosen to consider only the linear correlation measure. Additionally, we consider only the correlation between numerical covariates of the VAT data set in this preliminary descriptive analysis.

Identifying such groups for this particular data set is not as straight forward as one might first think. This is due to the presence of missing values in the data. In fact, in the VAT data set there are no observations without any missing values for all covariates, and the number of observations is more than halved if we demand that only the first ten covariates should contain no missing values. Computing a correlation matrix for all 500 covariates is thus not possible, and a sequential approach must be taken instead. Algorithm 5.1 was used to find groups of variables. The algorithm starts

Algorithm 5.1 Pseudocode for finding groups.

```
candidates = 1:ncol(X)#vector of covariate indexes
groups = list() # object of type 'list'
index_group = 1
For i in 1:ncol(X) do
    If i in candidates Then
        Add variable nr. i to groups[index_group]
        For j in candidates, and j!=i do
            If Cor(X[,i],X[,j]) > τ Then
                Add variable j to groups at index_group
            End If
        End For
        corMat = Cor(X[,groups[index_group]]) # corr. matrix.
        If any elements in corMat < τ Then
            Remove from groups[index_group] s.t.
            all correlations are >= τ
        End If
    End If
    remove variables in groups[index_group] from candidates
    index_group = index_group + 1
End For
Return groups
```

with the first covariate, adds the second covariate and checks if all elements in their correlation matrix are of the same sign, and that the correlations in absolute value are greater than τ. If this is the case, the covariate is kept, and if not the covariate is thrown away. We then add the next covariate to the correlation matrix and repeat the process. By iterating through all the remaining covariates we have thus found our first group. The second iteration starts with the first variable which was not included in the first group, and repeats the procedure of computing correlation matrices and checking the conditions. Note that Algorithm 5.1 is just a pseudocode, but

the overall logic is consistent with that of the real implementation. The only difference is in several conditions that need to be checked. For instance the pseudocode assumes no missing values, but the exact details of the algorithm are not of primary interest. Algorithm 5.1 does not seek to find the absolute best group compositions (however measured), but simply generates one set of possible groups. This is however not a large concern, since the problem at hand is to see whether placing covariates in groups of variables is at all a reasonable interpretation of the covariate structure.

The results of applying Algorithm 5.1 to the tax data set are shown in Table 5.1. Group sizes are given in the top row, with the number of groups for three different values of τ in the rows below.

	Group size										
τ	1	2	3	4	5	6	7	8	9	11	18
0.5	197	62	21	8	6	5	1	1	2	1	1
0.7	252	67	15	5	2	4	1	1	1	1	1
0.9	360	55	7	3	3	2	0	1	0	0	0

Table 5.1: Number of groups of different sizes for three lower correlation thresholds τ: 0.5, 0.7 and 0.9.

For a threshold of $\tau = 0.5$ the largest group found was of size 18, with 1 group of size 11 and 2 groups of size 9, in addition to many smaller groups. Note that only 197, less than half of the covariates could not be placed in a group. By increasing τ to 0.7 the groups generally decrease in both size and numbers. The largest group is still of size 18 as before, but there is only 1 group of size 9, and fewer groups of sizes 6 through 3. The number of variables not placed in a group has increased from 197 to 252. Lastly, increasing τ to 0.9 reduces the largest group size from 18 to 8, with a total of 360 variables that could not be placed in a group. In addition to finding such groups, we would also like to estimate the correlation both within, and between such groups of covariates. Presenting the correlations between all variables in all groups is not only impratical due to the sheer size of the resulting matrix, but is not very informative either. For these reasons we will instead compute the average correlation between two groups of covariates. For presentation purposes let the 10 largest groups of covariates be indexed according to the group size in descending order. Thus for $\tau = 0.5$, group 1 is the group with 18 covariates, group 2 with 11 covariates, and so on. Define the index set I_j containing the indexes of the variables contained in

group j. The matrix containing the average group correlations between the 10 groups is then given by

$$\boldsymbol{\rho}_{\text{group}} = \begin{pmatrix} \bar{\rho}_{1,1} & \bar{\rho}_{1,2} & \bar{\rho}_{1,3} & \cdots & \bar{\rho}_{1,10} \\ \bar{\rho}_{2,1} & \bar{\rho}_{2,2} & \bar{\rho}_{2,3} & \cdots & \bar{\rho}_{2,10} \\ \bar{\rho}_{3,1} & \bar{\rho}_{3,2} & \bar{\rho}_{3,3} & \cdots & \bar{\rho}_{3,10} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \bar{\rho}_{10,1} & \bar{\rho}_{10,2} & \bar{\rho}_{10,3} & \cdots & \bar{\rho}_{10,10} \end{pmatrix},$$

with $\bar{\rho}_{i,j}$ defined as

$$\bar{\rho}_{i,j} = \frac{1}{N} \sum_{i^* \in \mathrm{I}_i} \sum_{j^* \in \mathrm{I}_j : j^* \neq i^*} \mathrm{Cor}(\mathrm{x}_{i^*}, \mathrm{x}_{j^*}),$$

for $i, j = 1, \ldots, 10$, where N is the number of elements in this double sum making $\bar{\rho}_{i,j}$ an average. Consider first the scenario when the lower correlation threshold is $\tau = 0.5$. Computing the group average correlations yields the matrix

$$\hat{\boldsymbol{\rho}}_{\text{group}} = \begin{pmatrix} \mathbf{0.95} & 0.07 & 0.02 & 0.02 & 0.01 & 0.01 & \mathbf{-0.95} & 0.02 & 0.01 & -0.01 \\ & \mathbf{0.79} & 0.02 & \mathbf{0.12} & 0.03 & 0.05 & -0.1 & -0.01 & -0.01 & 0.01 \\ & & \mathbf{0.88} & 0.01 & 0.00 & 0.00 & -0.03 & 0.00 & 0.00 & 0.00 \\ & & & \mathbf{0.87} & 0.00 & 0.00 & -0.04 & -0.02 & 0.00 & 0.00 \\ & & & & \mathbf{0.81} & -0.03 & -0.02 & 0.00 & -0.01 & 0.00 \\ & & & & & \mathbf{0.87} & -0.03 & 0.00 & 0.00 & 0.00 \\ & & & & & & \mathbf{0.93} & 0.00 & 0.00 & 0.01 \\ & & & & & & & \mathbf{0.70} & 0.00 & 0.01 \\ & & & & & & & & \mathbf{0.93} & 0.00 \\ & & & & & & & & & \mathbf{0.96} \end{pmatrix}.$$

The lower triangle of $\hat{\boldsymbol{\rho}}_{\text{group}}$ is not shown, and elements greater than or equal to 0.1 are highlighted to improve readability. The first row contains average group correlations between group 1 and all other groups, row 2 contains average group correlations between group 2 and all other groups, and so on. We can see that average within-group correlations are generally quite large, with all of them being 0.7 or greater. With a lower correlation threshold of $\tau = 0.5$, inter-group dependence is not present to a very large degree. We run the same computations again, but now with a correlation threshold of $\tau = 0.9$. The groups are still indexed in descending order according to their size, as given in Table 5.1. The average group correlations are now given by

$$\hat{\boldsymbol{\rho}}_{\text{group}} = \begin{pmatrix} \mathbf{0.96} & 0.05 & 0.00 & 0.00 & 0.00 & -0.01 & 0.00 & -0.02 & -0.04 & 0.00 \\ & \mathbf{0.96} & 0.01 & 0.01 & -0.01 & \mathbf{0.86} & \mathbf{-0.88} & -0.01 & 0.01 & 0.00 \\ & & \mathbf{0.93} & 0.00 & 0.00 & 0.02 & -0.03 & 0.00 & 0.03 & 0.00 \\ & & & \mathbf{0.95} & 0.00 & 0.01 & 0.00 & 0.00 & -0.01 & 0.00 \\ & & & & \mathbf{0.99} & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ & & & & & \mathbf{1.00} & \mathbf{-0.93} & 0.08 & \mathbf{0.20} & 0.00 \\ & & & & & & \mathbf{0.93} & -0.06 & \mathbf{-0.10} & 0.00 \\ & & & & & & & \mathbf{0.97} & \mathbf{0.21} & 0.00 \\ & & & & & & & & \mathbf{0.94} & 0.00 \\ & & & & & & & & & \mathbf{0.99} \end{pmatrix}.$$

We see that the average within-group correlation has increased when the lower correlation threshold was increased to $\tau = 0.9$. Perhaps more interesting is that inter-group correlation now is present to a larger degree. The reason for this is that the increase in the lower correlation threshold creates splits in several groups. These are groups of variables which may be highly correlated, but the correlation is not high enough to form a single group. We see from Table 5.1 that increasing the lower correlation from 0.7 to 0.9 caused a split of the groups of size 18 and 11 into smaller groups. The resulting smaller groups will then have a large inter-group correlation, which is seen in these results.

We have in this section studied the dynamics of the dependence structure in the covariates of the VAT data set from Chapter 1. Our simple descriptive analysis indicates that dependence between covariates should be taken into account when generating data for a fraud detection scenario. Further, it appears that structuring the covariates into groups based on their inward correlation is a good idea. Lastly, we observed that such groups of variables are not only inwardly correlated, but we also observed inter-group correlation. We will in the following sections define a stochastic model from which we can sample new data with such characteristics as those we have just observed. However, in order to do this, we first need to introduce the concept of a copula.

5.2 Copulas

A copula $C_{1,\dots,d}(u_1, \dots, u_d)$ is a d-dimensional cumulative distribution function where the marginals $u_1, \dots, u_d$ are themselves uniformly distributed on $(0, 1)$. An interesting property illustrating the versatility of copulas is given by Sklars theorem (Sklar, 1959). It states that any d-dimensional cumulative distribution function $F_{1,\dots,\mathrm{d}}(z_1, \dots, z_\mathrm{d})$ with marginal cumulative distribution functions $F_i(z_i)$ for $i = 1, \dots, d$ can be expressed as

$$F_{1,\dots,d}(z_1, \dots, z_d) = C_{1,\dots,d}(F_1(z_1), \dots, F_d(z_d)).$$

Further, if $z_1, \dots, z_d$ are all continuous, then $C_{1,\dots,d}(\cdot)$ is unique. Many different models for $C_{1,\dots,d}(u_1, \dots, u_d)$ exist, but perhaps the most used, which will also be used in this chapter, is the Gaussian copula.

5.2.1 The Gaussian copula

The Gaussian copula is an elliptical copula, and thus has a symmetric dependence structure. The dependence is described by a $d \times d$ correlation matrix $\boldsymbol{\rho}$. Being an implicit copula, it has no explicit expression for the copula function $C_{1,\ldots,d}(u_1, \ldots, u_d)$. However, if $\boldsymbol{\rho}$ is positive definite its density exists and is given by

$$c_{1,\ldots,d}(u_1, \ldots, u_d) = \frac{\exp\{-\frac{1}{2}\mathbf{z}^T\boldsymbol{\rho}^{-1}\mathbf{z}\}}{|\boldsymbol{\rho}|^{\frac{1}{2}} \exp\{-\frac{1}{2}\sum_{i=1}^{d} z_i^2\}}, \tag{5.1}$$

where $\mathbf{z} = (z_1, \ldots, z_d)^T = (\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_d))^T$, and $\Phi(z)$ is the cumulative distribution function for the standard normal distribution.

The d-dimensional Gaussian copula with a correlation matrix $\boldsymbol{\rho}$ is the dependence structure of a d-dimensional Gaussian distribution with correlation matrix $\boldsymbol{\rho}$. A possible sampling strategy for a Gaussian copula is therefore to sample $\mathbf{z} = (z_1, \ldots, z_d)^T$ from a d-dimensional Gaussian distribution, and then transform these into uniforms via $\Phi(z_i) = u_i$ for $i = 1, \ldots, d$. This approach may therefore be seen as the opposite of inversion sampling, a quite general and common sampling method given by Algorithm 5.2. In inversion sampling we draw uniforms $\mathbf{u} = (u_1, \ldots, u_d)^T$, and then apply the inverse cumulative distribution function $F^{-1}(\cdot)$ to obtain the desired variable $\mathbf{z}$. In Algorithm 5.4 we instead sample $\mathbf{z}$, in this case from a Gaussian distribution, and convert these data into uniforms $\mathbf{u}$.

Algorithm 5.2 Inversion sampling for a continuous distribution with inverse cumulative distribution function $F^{-1}(U)$

1: Draw U $\sim$ Uniform$(0, 1)$
2: $Z = F^{-1}(U)$
3: **Return** Z

Algorithm 5.3 Inversion sampling for Bernoulli(p) distribution

1: Draw U $\sim$ Uniform$(0, 1)$
2: **If** U$< 1 - p$ **Then**
3: $\quad Z = 0$
4: **Else**
5: $\quad Z = 1$
6: **End If**
7: **Return** Z

Algorithm 5.4 Sampling from a Gaussian copula

1: Draw $\mathbf{z} \sim \mathrm{N}(\mathbf{0}_d, \boldsymbol{\rho}_{d\times d})$
2: Set $u_i = \Phi(z_i)$ for $i = 1, \ldots, d$
3: **Return** $\mathbf{u} = (u_1, \ldots, u_d)^T$

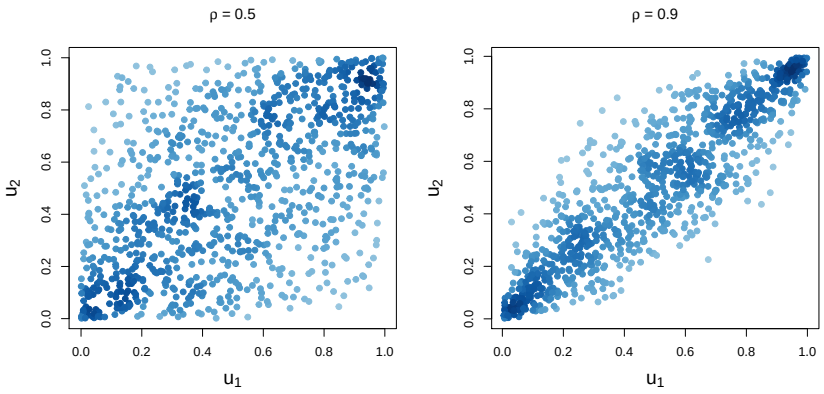


Figure 5.1: Data sampled from a bivariate Gaussian copula.

Once the d correlated uniforms $\mathbf{u}$ are obtained via Algorithm 5.4 we can then apply continuous marginal distributions from different families of distributions as in inversion sampling, i.e. Algorithm 5.2. For Bernoulli random variables the algorithm must be modified resulting in Algorithm 5.3. When generating our data set we will not sample from any other discrete distribution than the Bernoulli distribution. The reason for this being that any discrete variable with m levels will be converted into $m-1$ indicator (i.e. binary) variables when regression is performed anyways. Further, dependence between such multi-level discrete random variables can be represented via dependence between the indicator variables modeled via a copula. Figure 5.1 was created by sampling from a bivariate Gaussian copula via Algorithm 5.4, with $\rho = 0.5$ and 0.9 in the left and right plot, respectively. We can see a clearer structure as ρ increases from 0.5 to 0.9, illustrating the increase in dependence between u_1 and u_2. Applying inverse cumulative distribution functions $F_i^{-1}(\cdot)$ to u_i yields new variables $\mathrm{x}_j = F_j^{-1}(u_j)$ for $j = 1, 2$. We are free to apply whatever CDF's F_1 and F_2 we like, making this a versatile setup for generating dependent data. This has been done for two different sets of distributions in Figure 5.2. Figure 5.2 shows scatterplots of data sampled with x_1 normally distributed and x_2 Gamma distributed (top), and x_1 normally distributed but x_2 following a

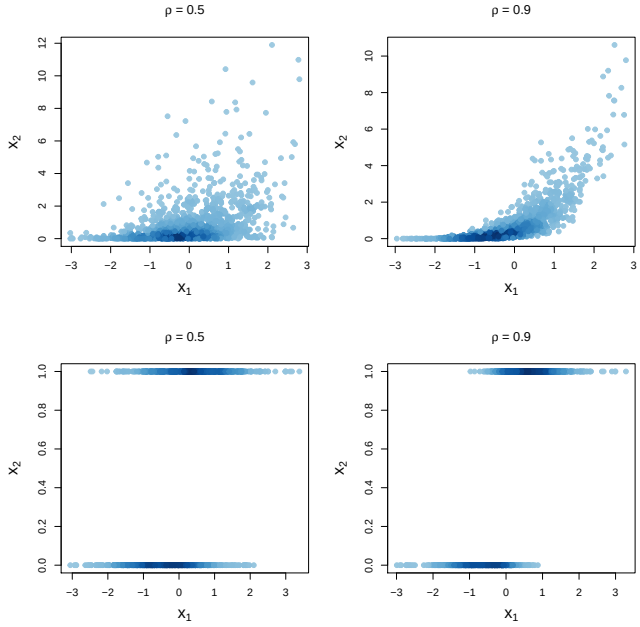


Figure 5.2: Data sampled via a bivariate Gaussian copula, applying different distributions to u_1 and u_2 in the generation of x_1 and x_2.

Bernoulli distribution (bottom). A bivariate Gaussian copula with $\rho = 0.5$ (first column) and $\rho = 0.9$ (second column) models the dependence (as in Figure 5.1) between x_1 and x_2. Note that the correlation matrix of our copula, $\boldsymbol{\rho}$, is in general not preserved under such transformations. For example, though the correlation between u_1 and u_2 is $\rho = 0.9$ in the top right plot of Figure 5.2, the correlation between x_1 and x_2 is not.

5.3 Generating a data set

Using the copula construction outlined in Section 5.2.1 we can generate data sets with similar properties as the VAT data set we analyzed in Section 5.1. This process can be split up in four main parts as illustrated by Figure 5.3.

Figure 5.3: Schematic illustratic the steps in generating a data set.

As we see from this setup, each part of the data generation procedure

is done independently of the others, so changing for instance the copula from which we draw $\mathbf{u}$, or the marginals for $\mathbf{x}_i$ can be done with ease. In this and subsequent sections the notation of the copula will be given in p dimensions, not d. This is because the goal is to generate covariates for a p-dimensional classification problem. We thus make the notation more situation specific. We begin with a closer look at the first box in Figure 5.3, namely drawing the uniforms $\mathbf{u}_i = (u_{i,1}, \ldots, u_{i,p})$. This requires that we first define the correlation matrix $\boldsymbol{\rho}$ for our copula.

5.3.1 Defining the correlation matrix

Defining the correlation matrix requires some thought regarding the structure and properties we would like our data set to represent. A starting point is to set $\text{Cor}(u_{i,j}, u_{i,k}) = \rho$, $\forall j \neq k$, leading to the correlation matrix

$$\boldsymbol{\rho} = \begin{pmatrix} 1 & \rho & \rho & \cdots & \rho \\ \rho & 1 & \rho & \cdots & \rho \\ \rho & \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \rho & \cdots & 1 \end{pmatrix}.$$

However, this may not give us the desired dependence structure as all $u_{i,1}, \ldots, u_{i,p}$ will now have an equal correlation. To produce an appropriate dependence structure we look to Section 5.1. We observed that one possible interpretation of the collinearity in our data is that the covariates can appear in groups. Further, we observed that such covariates were highly correlated with covariates in the same group. Our correlation matrix may then be given the block matrix representation

$$\boldsymbol{\rho} = \begin{pmatrix} \boldsymbol{\rho}_{1,1} & \boldsymbol{\rho}_{1,2} & \cdots & \boldsymbol{\rho}_{1,n_g} \\ \boldsymbol{\rho}_{2,1} & \boldsymbol{\rho}_{2,2} & \cdots & \boldsymbol{\rho}_{2,n_g} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\rho}_{n_g,1} & \boldsymbol{\rho}_{n_g,2} & \cdots & \boldsymbol{\rho}_{n_g,n_g} \end{pmatrix}, \tag{5.2}$$

where the matrices $\boldsymbol{\rho}_{i,j}$ are of dimension $p_g \times p_g$ for $i, j = 1, \ldots, n_g$. By introducing this structure for the correlation matrix $\boldsymbol{\rho}$ we make the simplifying assumption that all groups of covariates are of the same size p_g. Additionally, let us consider only three possibilities for $\boldsymbol{\rho}_{i,j}$, that is, three kinds of matrixes: $\boldsymbol{\rho}_h$,$\boldsymbol{\rho}_m$ and $\boldsymbol{\rho}_l$. These will be referred to as *high level*,

medium level and *low level* dependence matrices defined as

$$\boldsymbol{\rho}_h = \begin{pmatrix} 1 & \rho_h & \cdots & \rho_h \\ \rho_h & 1 & \cdots & \rho_h \\ \vdots & \vdots & \ddots & \vdots \\ \rho_h & \rho_h & \cdots & 1 \end{pmatrix}, \boldsymbol{\rho}_m = \begin{pmatrix} \rho_m & \rho_m & \cdots & \rho_m \\ \rho_m & \rho_m & \cdots & \rho_m \\ \vdots & \vdots & \ddots & \vdots \\ \rho_m & \rho_m & \cdots & \rho_m \end{pmatrix},$$

$$\boldsymbol{\rho}_l = \begin{pmatrix} \rho_l & \rho_l & \cdots & \rho_l \\ \rho_l & \rho_l & \cdots & \rho_l \\ \vdots & \vdots & \ddots & \vdots \\ \rho_l & \rho_l & \cdots & \rho_l \end{pmatrix},$$

respectively. The high level dependence matrix $\boldsymbol{\rho}_h$ has 1 in its diagonal and thus models dependence within one group of variables. The other matrices, the medium and low level dependence matrices $\boldsymbol{\rho}_m, \boldsymbol{\rho}_l$ will be used to model dependence between groups of variables. This gives us a structured, yet flexible model consistent with our observations made from real data. We can now let $\boldsymbol{\rho}$ take the form

$$\boldsymbol{\rho}_{i,j} = \begin{cases} \boldsymbol{\rho}_h & \text{if } |i-j| = 0 \\ \boldsymbol{\rho}_m & \text{if } |i-j| = 1 \\ \boldsymbol{\rho}_l & \text{if } |i-j| = 2 \\ \mathbf{0}_{p_g \times p_g} & \text{if } |i-j| > 2. \end{cases}$$

So within-group dependence is defined by the matrix $\boldsymbol{\rho}_h$, groups with a difference in indexes of 1 have a correlation given by $\boldsymbol{\rho}_m$, groups with a difference in indexes of 2 have a correlation matrix given by $\boldsymbol{\rho}_l$. For groups of variables where the difference in indexes is greater than 2 the correlation is 0. It is perhaps easier with a visual display of our final correlation matrix $\boldsymbol{\rho}$, which would be

$$\boldsymbol{\rho} = \begin{pmatrix} \boldsymbol{\rho}_h & \boldsymbol{\rho}_m & \boldsymbol{\rho}_l & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\rho}_m & \boldsymbol{\rho}_h & \boldsymbol{\rho}_m & \boldsymbol{\rho}_l & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\rho}_l & \boldsymbol{\rho}_m & \boldsymbol{\rho}_h & \boldsymbol{\rho}_m & \boldsymbol{\rho}_l & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\rho}_l & \boldsymbol{\rho}_m & \boldsymbol{\rho}_h & \boldsymbol{\rho}_m & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\rho}_l & \boldsymbol{\rho}_m & \boldsymbol{\rho}_h & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\rho}_h & \boldsymbol{\rho}_m & \boldsymbol{\rho}_l \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\rho}_m & \boldsymbol{\rho}_h & \boldsymbol{\rho}_m \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\rho}_l & \boldsymbol{\rho}_m & \boldsymbol{\rho}_h \end{pmatrix}.$$

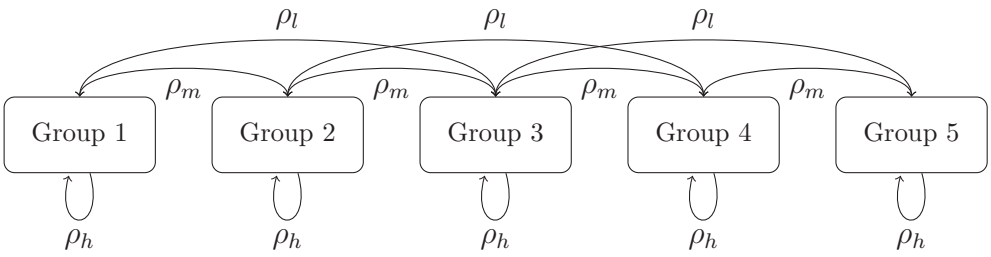


Figure 5.4: A Schematic illustrating the dependence structure in a 5-group scenario. Arrows imply dependence.

Such a construction allows for a dynamic in $\mathbf{u}_i = (u_{i,1}, \ldots, u_{i,p})$, so that the variables within a group are highly correlated, a lower correlation with variables within other groups, even allowing for independence between groups of variables. This is in accordance with the description of our data set in section 5.1. We illustrate the dependence structure by an example. Consider a scenario with 5 groups of covariates. The dependence structure is represented by Figure 5.4. For instance, we may omit the between-group dependence for groups with a distance of 2 by setting $\rho_l = 0$, or omitting the between-group dependence altogether by setting $\rho_m = \rho_l = 0$.

One downside with the current definition of $\boldsymbol{\rho}$ is that the correlation between variables in the same group is identical. This seems unrealistic, however we change this by adding weights $\exp\{-\omega(|i-j|-1)\}$ to the off-diagonal elements of $\boldsymbol{\rho}_h$. Let all off-diagonal elements in $\boldsymbol{\rho}_h$ take the form

$$\rho_h \exp\{-\omega(|i-j|-1)\},$$

with the constraint

$$\rho_h \exp\{-\omega(|i-j|-1)\} > c$$

for all $i, j = 1, \ldots, p_g$, which ensures that all correlations in a high level block are still greater than the some constant $c \geq \rho_m$. Solving the above inequality yields

$$\omega < \frac{\log(\rho_h/c)}{p_g - 2}, \tag{5.3}$$

since the maximum of $|i-j|-1$ is $p_g - 2$. The strength of dependence is thus affected by the distance (in indexes) between two variables within the same group. This does create quite a symmetric and structured dependence

structure which again may seem strange. However, one should emphasize this too much, since one can simply shuffle the covariates around, and the symmetry in $\boldsymbol{\rho}$ would disappear.

From our analysis of the VAT data set in Section 5.1 we can see that several variables were not placed in a group. Though this does not mean that variables which were not included in a group are completely independent of all other covariates, we make this simplifying assumption here. We will therefore introduce what we will denote *independence groups* which are groups where all covariates within this group are independent of each other and independent all other groups of covariates. For instance, by letting group number j be an independence group, we get

$$\begin{aligned}\boldsymbol{\rho}_{j,i} &= \boldsymbol{\rho}_{i,j} = \mathbf{0}_{p_g \times p_g} \text{ for } i = 1, \ldots, n_g, i \neq j, \\ \boldsymbol{\rho}_{j,j} &= \mathbf{I}_{p_g \times p_g}.\end{aligned}$$

Note that when constructing the correlation matrix ρ we are subject to the constraints following (5.1), namely that $\boldsymbol{\rho}$ must be **positive definite**.

5.3.2 Generating X

Now that we have defined our correlation matrix $\boldsymbol{\rho}$, we can draw the uniforms $\mathbf{u}_i$ using Algorithm 5.4. The next step is to transform these into our covariates $\mathbf{x}_i$. We will use the sampling techniques described at the end of the previous section for this, i.e. Algorithm 5.2 and 5.3. This allows for correlated explanatory variables, with a degree of dependence we can decide ourselves. It also means we can produce data sets with a combination of correlated discrete and continuous explanatory variables, as illustrated in the previous section. In mathematical terms, this translates to

$$\begin{aligned}\mathbf{u}_i &\sim C_{1,\ldots,p}(u_{i,1}, \ldots, u_{i,p}) \\ \mathbf{x}_i &= \left(F_1^{-1}(u_{i,1}), \ldots, F_p^{-1}(u_{i,p})\right),\end{aligned}$$

where $C_{1,\ldots,p}(\cdot)$ is the Gaussian copula. By combining Algorithm 5.4, 5.2 and 5.3 we get an algorithm for generating $\mathbf{X}$.

5.3.3 Generating Y

Before we can generate the response variable Y_i we first need to specify a model for $P(Y_i = 1|\mathbf{x}_i)$. Because this thesis focuses on logistic regression,

we will here use the model

$$P(Y_i = 1|\mathbf{x}_i) = \frac{\exp\{\mathbf{x}_i^T\boldsymbol{\beta}\}}{1 + \exp\{\mathbf{x}_i^T\boldsymbol{\beta}\}}.$$

This is then used to sample Y_i from a Bernoulli distribution with probability of success given by $P(Y_i = 1|\mathbf{x}_i)$. Thus, a prerequisite for generating Y_i is a proper choice of $\boldsymbol{\beta}$. We can decide to let $\boldsymbol{\beta}$ have many, but small coefficients, or only a few but rather large and important coefficients. Given our dependence structure for our covariates discussed earlier, it is also important which elements in $\boldsymbol{\beta}$ are set to non-zero. Setting only elements that are adjacent to each other to non-zero will result in a model where most covariates with non-zero coefficients are highly correlated. Alternatively, one could set the coefficients in $\boldsymbol{\beta}$ such that the covariates whose $\boldsymbol{\beta}$-coefficients are non-zero are placed in different groups. This would give less correlation between the covariates with non-zero coefficients. Arguably, it would make for a more realistic model, since it is rarely the case that all explanatory variables in a data set are highly correlated. Lastly, one could place the non-zero coefficients in $\boldsymbol{\beta}$ so far apart that their associated predictors become independent.

Chapter 6

Simulation study: regularization methods

In this chapter we study the behavior of the regularization methods from Chapter 3 in situations where covariates are highly dependent. This will be done by adopting the framework of the stochastic model outlined in Chapter 5. Parameters in the data generation model will be varied across simulations to recreate a range of different dependence structures between the covariates.

6.1 Experiment design

Generating a data set with a large degree of dependence requires some initial considerations to be made. This is partly due to the inherent flexibility in the model introduced in Chapter 5. For instance, the true model from which we generate data can be changed by both varying $\boldsymbol{\beta}$ or $\boldsymbol{\rho}$, and keeping in mind the combined effects of these parameters is important. In the following subsections, we will give the details of our different simulation experiments. There are however some parameters that will not change throughout these simulations. The number of covariates will be set to $p = 1000$, of which 100 will have $\boldsymbol{\beta}$- coefficients not equal zero. The value of the intercept β_0 is not of primary interest here, since no shrinkage is applied to it by any of the regularization methods we have discussed. However, the value of β_0 does impact the balance of our data set, and choosing an appropriate value for β_0 prevents any issues regarding class imbalance, which is a separate problem we will deal with later. Recall the logistic regression model

$$P(Y_i = 1|\mathbf{x}_i) = \frac{\exp\{\beta_0 + \mathrm{x}_{i,1}\beta_1 + \cdots + \mathrm{x}_{i,p}\beta_p\}}{1 + \exp\{\beta_0 + \mathrm{x}_{i,1}\beta_1 + \cdots + \mathrm{x}_{i,p}\beta_p\}},$$

and observe that if $\beta_0 + \mathrm{x}_{i,1}\beta_1 + \cdots + \mathrm{x}_{i,p}\beta_p = 0$, then $P(Y_i = 1|\mathbf{x}_i) = 0.5$. We can then set β_0 such that $E[\beta_0 + \mathrm{x}_{i,1}\beta_1 + \cdots + \mathrm{x}_{i,p}\beta_p] = 0$, which gives $\beta_0 = -\sum_{j=1}^{p} E[\mathrm{x}_{i,j}]\beta_j$. By using this value of β_0 we can ensure that our data sets are not unnecessarily unbalanced to consist of a majority of either 1's or 0's. This also ensures consistency between different simulation scenarios, since class imbalance will not be a contributing factor in our simulation results. If we did not ensure this, we could have ended up with values of β_0 that made our data sets be slightly skewed to contain more 0's in one scenario, and more 1's in another. While such a skew may not have been big, it could certainly have impacted the model estimation procedure and thus contributed to a distortion of our final results and conclusions. The correlation matrix $\boldsymbol{\rho}$ of the Gaussian copula $C_{1,\ldots,p}(u_1, \ldots, u_p)$ will be set as a block matrix as discussed in Chapter 5. The parameter values for ρ_h, ρ_m, ρ_l and ω will vary between different simulations, the details of which we will now provide.

6.1.1 Simulation 1: within-group dependence

In this first set of simulations, the number of observations n will be set to 3000. The group size p_g will be set to 10, such that the number of groups n_g is 100. We will in this scenario consider a model where the explanatory variables have a within-group dependence structure, but no inter-group dependence. Further, the within-group dependence will be constant. The values defining the correlation matrix are set to

$$\rho_h = 0.9, \quad \rho_m = 0, \quad \rho_l = 0, \quad \omega = 0.$$

As we mentioned in Chapter 5, some groups of covariates should be set independent of all other covariates, the so called independence groups. We define the last 10 groups of covariates to be just such independence groups. After $u_1, \ldots, u_{1000}$ have been drawn, we will transform these as described in Chapter 5. The first 3 variables in each group will be standard normally distributed, the next 3 will be Gamma(1,1) distributed and the last 4 of each group will be Bernoulli distributed with probability of success $p = 0.5$. Thus, in group $j = 1, \ldots, n_g$ we have

$$\begin{aligned}
X_{i,p_g(j-1)+k} &\sim \mathrm{N}(0,1) \text{ for } k = 1,2,3 \\
X_{i,p_g(j-1)+k} &\sim \mathrm{Gamma}(1,1) \text{ for } k = 4,5,6 \\
X_{i,p_g(j-1)+k} &\sim \mathrm{Bernoulli}(0.5) \text{ for } k = 7,8,9,10.
\end{aligned}$$

Group 1	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9	β_{10}
Group 2	β_{11}	β_{12}	β_{13}	β_{14}	β_{15}	β_{16}	β_{17}	β_{18}	β_{19}	β_{20}
⋮					⋮					
Group 15	β_{141}	β_{142}	β_{143}	β_{144}	β_{145}	β_{146}	β_{147}	β_{148}	β_{149}	β_{150}
Group 16	β_{151}	β_{152}	β_{153}	β_{154}	β_{155}	β_{156}	β_{157}	β_{158}	β_{159}	β_{160}
Group 17	β_{161}	β_{162}	β_{163}	β_{164}	β_{165}	β_{166}	β_{167}	β_{168}	β_{169}	β_{170}
Group 18	β_{171}	β_{172}	β_{173}	β_{174}	β_{175}	β_{176}	β_{177}	β_{178}	β_{179}	β_{180}
⋮					⋮					
Group 95	β_{941}	β_{942}	β_{943}	β_{944}	β_{945}	β_{946}	β_{947}	β_{948}	β_{949}	β_{950}
Group 96	β_{951}	β_{952}	β_{953}	β_{954}	β_{955}	β_{956}	β_{957}	β_{958}	β_{959}	β_{960}
Group 97	β_{961}	β_{962}	β_{963}	β_{964}	β_{965}	β_{966}	β_{967}	β_{968}	β_{969}	β_{970}
Group 98	β_{971}	β_{972}	β_{973}	β_{974}	β_{975}	β_{976}	β_{977}	β_{978}	β_{979}	β_{980}
Group 99	β_{981}	β_{982}	β_{983}	β_{984}	β_{985}	β_{986}	β_{987}	β_{988}	β_{989}	β_{990}
Group 100	β_{991}	β_{992}	β_{993}	β_{994}	β_{995}	β_{996}	β_{997}	β_{998}	β_{999}	β_{1000}

Figure 6.1: Illustration of $\boldsymbol{\beta}$-coefficients. Grey color indicates non-zero value for the relevant coefficient.

Since explanatory variables in all groups are transformed this way, we will have a data set where 60% of the explanatory variables are continuous, and 40% are binary.

We will define our $\boldsymbol{\beta}$ such that there are 5 non-zero coefficients for selected groups of variables. Every 1st, 3rd, 5th, 7th and 9th β-values in these groups will be set to 0.05. These groups are the first 16, and the last 4. In mathematical terms $\beta_{p_g(j-1)+k} = 0.05$, for $j = 1, \ldots, 16, 97, \ldots, 100$ and $k = 1, 3, 5, 7, 9$. Thus we have 16 groups where 5 of the covariates in each

group have non-zero coefficients in $\boldsymbol{\beta}$. In addition to this, we set 5 covariates in each of the 4 independence groups to have non-zero coefficients in $\boldsymbol{\beta}$. See Figure 6.1 for an illustration of which $\boldsymbol{\beta}$ coefficients are set as non-zero.

6.1.2 Simulation 2: varying the within-group dependence

We now change the dependence structure of our correlation matrix $\boldsymbol{\rho}$ such that the degree of within-group dependence depends on the distance between two covariates in the same group, as explained in Chapter 5. To ensure our results in this subsection will be comparable to those in the next subsections, we must consider what the value of ω should be. The value for ω is set such that the lowest level of within-group dependence would be 0.7, if the group size was 20. The reason for this is that we will in later simulation experiments increase the group sizes from 10 to 20. By setting such a value for now, ω we ensure that the dependence structure is identical for the first 10 covariates, independent of group size. Keeping in mind the constraint given in (5.3), the parameters defining $\boldsymbol{\rho}$ are now

$$\rho_h = 0.9, \quad \rho_m = 0, \quad \rho_l = 0, \quad \omega = 0.99\frac{\log(0.7/\rho_h)}{18}.$$

With these values, the lowest level of within group dependence is approximately 0.8. The $\boldsymbol{\beta}$-vector is defined as in Section 6.1.1.

6.1.3 Simulation 3: adding inter-group dependence

The third set of simulations rely on the setup from Section 6.1.2, with the only difference being that we add inter-group dependence. We observed such inter-group dependence between the groups of covariates in the VAT data set in Chapter 5.1, and we now wish to recreate this. Thus, the parameters defining $\boldsymbol{\rho}$ are now set to

$$\rho_h = 0.9, \quad \rho_m = 0.4, \quad \rho_l = 0.2, \quad \omega = 0.99\frac{\log(0.7/\rho_h)}{18}.$$

6.1.4 Simulation 4: spreading non-zero coefficients in $\boldsymbol{\beta}$

In the previous scenarios we have altered only the correlation matrix of our Gaussian copula. The difference now is in the $\boldsymbol{\beta}$-vector, which is defined such that every other group has covariates with non-zero coefficients

in $\boldsymbol{\beta}$. Again, more mathematically we have $\beta_{p_g(j-1)+k} = 0.05$, for $j = 1, 3, 5, 7, \ldots, 31, 97, \ldots, 100$ and $k = 1, 3, 5, 7, 9$. An interpretation for this setup is that we assume that groups of variables that do not contribute to the response are to a greater extent correlated with groups that have covariates that do impact the response. For the sake of an example, consider the covariate groups in Section 5.1. The scenarios in Sections 6.1.1,6.1.2 and 6.1.3 correspond to assuming that group number 2 and 5 in the second $\hat{\boldsymbol{\rho}}_{\text{group}}$ matrix of Section 5.1 both contribute to the response. We imagine now in this scenario that the correlation matrix is just as before, but group 5 no longer truly contributes to the response. Instead, say group 2 and 6, which have a lower inter-group correlation truly have an effect on the response. Groups 2 and 5 are still highly correlated, but no true effects on the response can be ascribed to group 5.

6.1.5 Simulation 5: change β coefficients

We continue to alter $\boldsymbol{\beta}$ in this scenario as we did in Section 6.1.4. As opposed to Section 6.1.4 we will not change the indexes of which $\boldsymbol{\beta}$-coefficients are non-zero but change their values. In order to introduce some variation in the $\boldsymbol{\beta}$ vector, we set the first 5 non-zero elements of $\boldsymbol{\beta}$ to 0.5, the next 5 non-zero elements to 0.1. We also introduce some larger coefficients, but spread these out such that they do not appear in the same groups of covariates. Specifically, we set the 20'th and 40'th non-zero $\boldsymbol{\beta}$-coefficients equal to 1, and the 60'th and 80'th non-zero $\boldsymbol{\beta}$-coefficients equal to 2. The remaining 85 non-zero $\boldsymbol{\beta}$-coefficients are set to 0.025. Down-scaling some of the coefficients is necessary in order to keep the signal to noise ratio at a reasonable level.

6.1.6 Simulation 6: increase group size to 20

Looking at the groups we obtained from the VAT data set in Section 5.1, we see that the largest group size was 18, while we so far in our simulation study have only considered groups of size 10. We wish to study impact of larger group sizes is this section, and thus we increase the group size p_g from 10 to 20, reducing the number of groups n_g to 50. The independence groups will now be set as the last 5 groups of covariates, to keep the total number of independent covariates the same as before. For the same reason that we had to be careful when defining ω in section 6.1.2, we now too must be careful in how we generate the groups of size 20. Given the correlation matrix, sampling from the Gaussian copula is done as before. However, it is important that the properties of the linear predictor $\eta(\mathbf{x}_i)$ are the same.

The marginal distributions will therefore be defined as

$$X_{i,p_g(j-1)+k} \sim \mathrm{N}(0,1) \text{ for } k = 1,\ldots,9$$
$$X_{i,p_g(j-1)+k} \sim \mathrm{Gamma}(1,1) \text{ for } k = 10,11,12$$
$$X_{i,p_g(j-1)+k} \sim \mathrm{Bernoulli}(0.5) \text{ for } k = 13,\ldots,20.$$

If we let the non-zero elements be the 7th, 9th, 11th, 13th and 15th covariates in each group, the properties of $\eta(\mathbf{x}_i)$ have not changed from section 6.1.2. Thus, the true model is exactly the same as before, but we have made the groups of variables larger, in the sense of adding more highly correlated "noisy" covariates.

6.1.7 Simulation 7: increase n

Finally in the last set of simulations we wish to study the impact of a larger data set. This is partly due to the observation that the VAT data set has 50255 observations. Another reason is that we wish to see if the oracle properties of SCAD and adaptive lasso make these methods benefit more from an increase in sample size. The sample size n will thus be increased by a factor of 10, from 3000 to 30000. All other details regarding the model from which we sample data are exactly as in Section 6.1.6.

6.2 Implementation and estimation

Setting up simulation experiments such as those described requires programming, in this case in R. The pseudocode for running these simulations is given in Algorithm 6.1.

Algorithm 6.1 Pseudocode for simlation experiments

Input: $method, m,$
Results = 1:m*0
For i in 1:m **do**
    Generate training data T_1 # Algorithms 5.4,5.2,5.3
    Generate test data T_2
    Train model on T_1
    Create test summaries for model, based on T_2#AUC,BS etc.
    Results[i] = test summaries #Based on step 6
End For
Return Results

The inputs of Algorithm 6.1 are the number of simulations m and the parameter *method* which denotes which of the estimation methods presented in Chapter 3 should be used for these simulations.

We wish to run each of our simulations 200 times for each scenario in order to get reliable results. However, for this to be possible in practice, we need our R code to be optimized. Of particular concern initially was the long execution times of generating the data set in steps 3 and 4 in Algorithm 6.1. This was much due to the fact that the R code was written to a large degree using for-loops, with little regards to optimization of execution times, but rather making the code intuitive and easy to understand and work with. Some time was therefore spent on trying to vectorize this particular part, which made a substantial impact on execution times. After the R-script had been improved with regards to computational efficiency, the runtime was substantially reduced. For instance, generating $n = 3000$ observations took about 25 minutes prior to vectorization, and roughly 15 seconds after vectorization. However, running these simulations would still take a long time. The next step was therefore to run the R script in **parallel**. More specifically, the for-loop in Algorithm 6.1 was implemented in parallel, rather than running it sequentially. Parallelizing in such situations is typically beneficial since each iteration runs independently of all the other iterations. Total execution time should therefore reduce in a linear fashion when the number of workers increases, which is considered to be an ideal situation in parallel programming. The package `doParallel` was used to run the for-loop in Algorithm 6.1 in R. A prerequisite for improvements in execution times when doing parallelized computing is adequate hardware. To overcome this potential limitation, the simulations were run on the **Abacus machine** at the University of Oslo which has a total of 48 cores.

Cross validation

All of the methods introduced in Chapter 3 require optimization over one or more parameters (hyper-parameters), with the exception of the maximum likelihood estimator. This will be done by 10-fold cross validation. More specifically, we will use a stratified cross validation approach, which we commented on in Chapter 2. We will use the AUC computed on the hold-out set as the measurement of model performance in our cross validation procedure. The value of the hyper-parameters that maximize the mean AUC across all folds is then chosen as the final hyper-parameter. Figure 6.2 shows AUC estimated by cross validation for different values of λ using lasso with logistic regression.

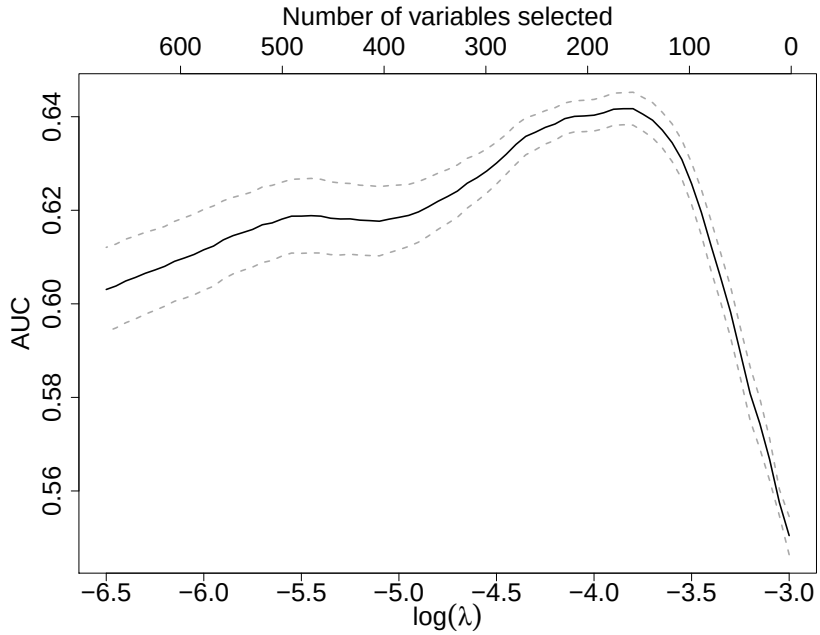


Figure 6.2: Mean cross validated AUC plus/minus one standard deviation.

For the elastic net and adaptive lasso the cross validation procedures are somewhat more complicated. Elastic net requires that we run the cross validation procedure as described above for a set of selected α values. Thus, if we wish to optimize the elastic net over $\alpha = 0.1, 0.5, 0.9$, we would run one cross validation procedure for each of these values of α, and record the best pair of (λ, α) to be used for our final model. The same procedure is used when optimizing over (λ, γ) for the adaptive lasso. However, there is one key aspect in which a cross validation procedure differs for adaptive lasso compared to elastic net. When optimizing the adaptive lasso over (λ, γ) we must be careful not to cause the model to overfit. This will occur if one computes weights $\hat{\text{w}}_j$ for $j = 1, \ldots, p$ based on all training data prior to the cross validation procedure, and then use these pre-estimated weights in all cross-validation folds. By doing this we first use all the training data to create the weights. We thus help the cross validation procedure somewhat, by providing weights that were computed on all training data. This will cause inclusion of too many parameters, and a bias in the estimated predictive performance of our model. The correct way is to compute the weights separately in each cross validation iteration to prevent over-fitting from occurring. This is however more computationally intensive.

Lasso and ridge

Both lasso and ridge can be fitted by cross-validation using the `glmnet` package. We will used Algorithm 2.1 to generate the cross validation folds.

Elastic net

As noted above, training the elastic net requires optimizing the parameters (α, λ). One option would be to set α as a fixed value, say 0.5, and then only optimize over λ. However, this is not a very flexible approach. We will instead run cross-validation over a grid of α and λ values as in Zou and Hastie (2005). This approach does have the disadvantage of increased computational cost. Thus, we will optimize over a grid that is not too high-resolution. Seeing that if $\alpha = 0$ we have the ridge penalty, and if $\alpha = 1$ we have the lasso penalty, we would like our α grid not to cover these values, since these penalty terms are tested separately. The grid of α values will be set to $(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99)$. We focus on the values close to 0 and 1 because the elastic net tends to have its optimal solution close to either a ridge ($\alpha = 0$) or lasso solution ($\alpha = 1$) as noted in Zou and Hastie, 2005.

Adaptive lasso

The adaptive lasso relies on a clever choice of the weights $\hat{\mathrm{w}}_j$. We will obtain the weights by $\hat{\mathrm{w}}_j = 1/\hat{\beta}_j^*$ for $j = 1, \ldots, p$, where $\hat{\beta}_j^*$ is the ridge estimator. Using the ridge estimator to compute the initial weights has been recommended in situations where collinearity is present, since it tends to be more stable than the maximum likelihood estimator in this case (Zou, 2006). To reduce computational cost, we will perform 5-fold cross validation for computing the ridge estimators to be used in the weights. Concerning the additional hyper-parameter γ, we will not run any cross validation over this parameter, but set it equal to $\gamma = 1$ for all runs. Initially, we tried running a two-dimensional cross-validation over the values $\gamma = 0.5, 1, 2$ as in Zou (2006), but the computational costs were far too great to run this for all scenarios. Additionally, when running these simulations on a smaller scale the values $\gamma = 0.5$ and $\gamma = 1$ tended to be selected most of the time, and thus we run only for $\gamma = 1$. Even with these simplifications to the training of the adaptive lasso, it still takes by far the most time to estimate.

SCAD

Fitting the logistic regression model with the SCAD penalization term in R can be done through the package `ncvreg`. A model can be fit for a specified value of λ, or one can optimize λ using the built-in cross validation procedures for this package. In the case of binary classification, the `ncvreg` package supports only binomial deviance computed on the hold-out set in cross validation as the measurement of model performance, so if we wish to use AUC as a performance measure we have to program such a routine. Regarding the a parameter, both empirical and theoretical arguments have been given to show that the SCAD penalty estimator is optimal around $a = 3.7$, Fan and Li (2001). We will therefore set $a = 3.7$ in the following simulations.

6.3 Model evaluation criteria

We are now in a setting where we know the true model for $P(Y_i = 1|\mathbf{x}_i)$, and this brings with it some advantages when it comes to contrasting and comparing our methods for model regularization. We want to consider the number of non-zero elements $N_{\hat{\beta}\neq 0}$ in $\widehat{\boldsymbol{\beta}}$, given by

$$N_{\hat{\beta}\neq 0} = \sum_{j=1}^{p} I(\hat{\beta}_j \neq 0).$$

In addition to this, we wish to know how often our estimated model correctly recognizes zero and non-zero coefficients in $\boldsymbol{\beta}$. Define the proportion of non-zero coefficients in $\boldsymbol{\beta}$ that were correctly estimated to non-zero $P_{\hat{\beta}\neq 0}$ and the proportion of zero-coefficients in $\boldsymbol{\beta}$ that were correctly estimated zero $P_{\hat{\beta}=0}$ as

$$P_{\hat{\beta}\neq 0} = \frac{\sum_{j:\beta_j\neq 0} I(\hat{\beta}_j \neq 0)}{\sum_{j=1}^{p} I(\beta_j \neq 0)}$$
$$P_{\hat{\beta}=0} = \frac{\sum_{j:\beta_j=0} I(\hat{\beta}_j = 0)}{\sum_{j=1}^{p} I(\beta_j = 0)},$$

respectively. A method which perfectly identifies both zero and non-zero coefficients therefore has $P_{\hat{\beta}=0} = P_{\hat{\beta}\neq 0} = 1$. To supplement these two measures we also consider the proportion of non-zero coefficients in $\hat{\boldsymbol{\beta}}$ that are actually non-zero, i.e. the corresponding coefficient in $\boldsymbol{\beta}$ is non-zero,

$$\text{ACC}_{\beta} = \frac{\sum_{j:\hat{\beta}_j\neq 0} I(\beta_j \neq 0)}{\sum_{j=1}^{p} I(\hat{\beta}_j \neq 0)}.$$

This can be considered a measure of the accuracy of the chosen covariates of a method. Due to this interpretation we will denote this by ACC_β. This can be equally informative as $P_{\hat{\beta}\neq 0}$ and $P_{\hat{\beta}=0}$ when considering the different estimation methods.

For evaluation of predictive performance we will report the AUC. We will also report the Brier score because the AUC has its limitations, as commented in Chapter 2. An optimal method in terms of predictive performance therefore scores the best in terms of both AUC and Brier score. Note that since we are optimizing our models w.r.t. AUC, the Brier score will serve as a secondary measure, since these models have not been fitted to optimize Brier score. The results would very likely have been different if we were to optimize with respect to Brier score, and report the AUC as a secondary predictive performance measure. These measures of predictive performance will be computed on an independent test set. This test set will be of size $n = 3000$ in all the following simulations.

6.4 Results

6.4.1 Simulation 1: within-group dependence

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.586	0.336
Ridge	1000	0	1	0.1	0.660	0.238
Lasso	74.2 (20.8)	0.953	0.315	0.425	0.661	0.231
SCAD	65.9 (21.9)	0.954	0.247	0.374	0.657	0.232
Elnet (0.03)	261.4 (75.9)	0.799	0.800	0.306	0.670	0.234
Alasso	110.5 (69.6)	0.918	0.365	0.330	0.654	0.233

Table 6.1: Simulation results for only within-group dependence (simulation 1), with $m = 200$ runs for each method.

The results from the simulation setup in Section 6.1.1 are given in Table 6.1. The penalty is given in the leftmost column, with "No" meaning no penalty was applied, and thus shows the results for a model fitted using maximum likelihood. The second column shows the average number of covariates estimated as non-zero (with standard deviation in parenthesis).

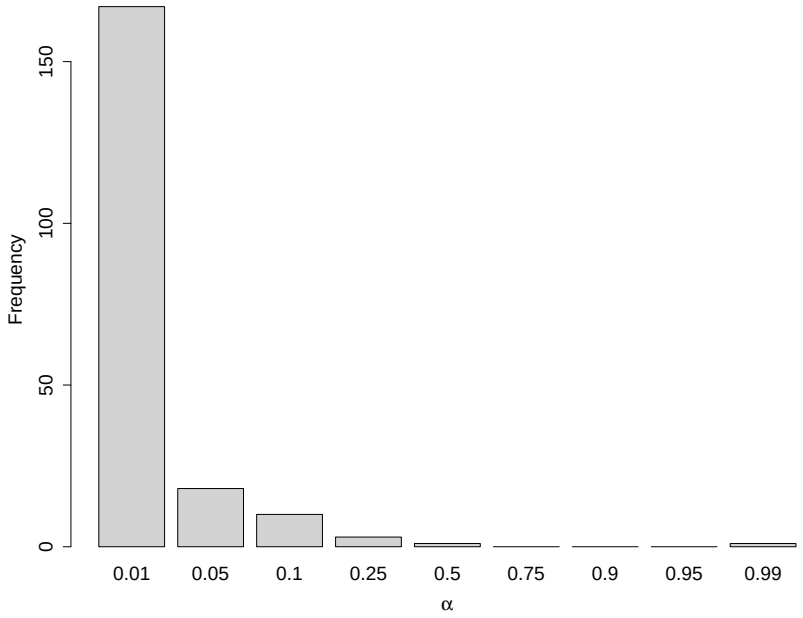


Figure 6.3: Distribution of α for elastic net in Simulation 1.

The maximum likelihood estimator as well as ridge does not perform variable selection, so this will always be 1000 for these methods. Of those methods that perform variable selection, elastic net by far keeps the largest number of variables with an average 261.4. This can partly be explained by the fact that the average α value was 0.03, which is close to a ridge penalty. Figure 6.3 shows how the α values are distributed in the simulations using elastic net. As we can see, the majority of α values are 0.01. The elastic net kept more than twice as many covariates as adaptive lasso, which averaged 110.5 non-zero coefficients. Both elastic net and adaptive lasso have quite large variability in the number of variables selected. The SCAD penalty seems to be the most conservative of the methods with an average of 65.9 variables selected. The lasso, with an average of 74.2 variables selected, is higher than SCAD, though fewer than adaptive lasso and elastic net. The second column contains values of $P_{\hat{\beta}=0}$, representing the proportion of zero coefficients that were correctly estimated as zero. These values must be seen in relation with $P_{\hat{\beta}\neq 0}$, which tends to mirror $P_{\hat{\beta}=0}$ since a large value of the former usually means a low value for the latter. To supplement these, we also provide the probability that a coefficient that has been estimated to be non-zero is actually non-zero, which we called the accuracy ACC_β. We see that lasso performs better than SCAD in terms of variable selection accuracy,

with both higher values of $P_{\hat{\beta}\neq 0}$ and ACC_β, and performing similarly with respect to $P_{\hat{\beta}=0}$. The adaptive lasso selects more variables, and does succeed in finding more of the truly non-zero coefficients than the lasso, which can be seen by a larger $P_{\hat{\beta}\neq 0}$. However, the adaptive lasso also erroneously selects many variables, giving it a lower ACC_β than lasso. Elastic net gives a quite different solution than any of the other methods, as can be seen by a substantially larger $P_{\hat{\beta}\neq 0}$. It includes roughly 80 % of all variables with non-zero coefficients. However, as with adaptive lasso, elastic net erroneously includes many covariates, resulting in a lower ACC_β than lasso.

In terms of predictive performance, elastic net performs best of all the methods, with an average AUC of 0.67. Ridge and lasso perform similarly, with AUC's of 0.660 and 0.661, respectively. SCAD and adaptive lasso perform slightly worse with AUC's of 0.657 and 0.654. We also see that all of the regularization methods perform better than the maximum likelihood fit, which achieves an AUC of 0.586, slightly better than chance. The ML fit performs worst of all models in terms of Brier score as well. Lasso performs best in this regard, though not by much when compared to most of the other regularization methods.

6.4.2 Simulation 2: varying the within-group dependence

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.579	0.338
Ridge	1000	0	1	0.1	0.658	0.238
Lasso	74.9 (30.4)	0.952	0.321	0.429	0.659	0.232
SCAD	65.3 (23.3)	0.955	0.248	0.380	0.653	0.233
Elnet (0.04)	266.9 (77.5)	0.793	0.802	0.300	0.667	0.234
Alasso	113.9 (88.6)	0.915	0.374	0.329	0.650	0.235

Table 6.2: Simulation results from introducing decaying within-group dependence (simulation 2), with $m = 200$ runs for each method.

The results from the simulation setup in this section are not much different from those of Section 6.1.1. This may be because the two models from which we sample our data are not that different. In terms of number of variables selected $N_{\hat{\beta}\neq 0}$, there are some minor differences for some of the methods,

which may well be due to randomness. Some, but minor differences can also be seen in $P_{\hat{\beta}=0}$ and $P_{\hat{\beta}\neq 0}$. Perhaps more interestingly is that the AUC has decreased slightly for all methods. While at this stage one might easily write this off as nothing more than randomness, this is consistent with a pattern which we will see more of in the results to come.

6.4.3 Simulation 3: adding inter-group dependence

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.632	0.321
Ridge	1000	0	1	0.1	0.740	0.237
Lasso	60.8 (15.4)	0.968	0.321	0.529	0.737	0.209
SCAD	44.9 (12.9)	0.979	0.259	0.577	0.733	0.212
Elnet (0.02)	222.7 (63.4)	0.844	0.821	0.369	0.743	0.213
Alasso	80.6 (61.7)	0.952	0.371	0.460	0.733	0.210

Table 6.3: Simulation results when introucing dependence between groups (simulation 3), with $m = 200$ runs for each method.

Whereas we saw little difference between results in Section 6.4.1 and 6.4.2, we now observe a more different set of results. The number of variables selected $N_{\hat{\beta}\neq 0}$ has decreased for all methods which leads to an increase in $P_{\hat{\beta}=0}$ for all methods. Lasso, for instance, is known to have this behavior, namely that it selects only a few of many highly correlated covariates. We see that for SCAD and elastic net, the proportion of non-zero coefficients correctly estimated as non-zero $P_{\hat{\beta}\neq 0}$ has also increased. All methods saw an increase in ACC_β.

As in the previous simulation experiments, elastic net performs best in prediction, with the highest AUC of 0.743. The differences in AUC compared to the other methods is small, especially ridge. It is interesting to see that when we increased the correlation between the coefficients, the AUC has increased for all methods. For maximum likelihood estimation, it increased from 0.579 to 0.632, while the regularized models saw an increase form approximately 0.65 to 0.74. Thus, it seems that as the strength of dependence between covariates increases, the AUC increases. Similarly, we also note that the Brier score has decreased. Thus, the task of classifying Y

has become easier as a result of increased correlation between the covariates. We will take a closer look at this behavior at the end of this chapter.

6.4.4 Simulation 4: spreading non-zero coefficients in $\boldsymbol{\beta}$

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.600	0.330
Ridge	1000	0	1	0.1	0.692	0.226
Lasso	70.8 (18.9)	0.957	0.317	0.448	0.692	0.223
SCAD	57.1 (17.3)	0.965	0.258	0.452	0.687	0.225
Elnet (0.03)	278.6 (89.3)	0.778	0.789	0.283	0.697	0.226
Alasso	107.3 (84.2)	0.920	0.351	0.327	0.681	0.227

Table 6.4: Simulation results for spreading non-zero coefficients in $\boldsymbol{\beta}$ (simulation 4), with $m = 200$ runs for each method.

The number of variables selected $N_{\hat{\beta}\neq 0}$ has now increased for all methods in comparison with section 6.4.3. Most notably, the elastic net now included on average 278.6 variables in the final model, which is more than in any of the previous sections. The elastic net is still close to a ridge solution, with an average α of 0.034. The inclusion of more variables makes both $P_{\hat{\beta}=0}$ and $P_{\hat{\beta}\neq 0}$ drop for all models, which when combined with an increase in the number of variables selected leads to lower ACC_β. This increase in the number of covariates selected, as well as the decrease in accuracy of these selected covariates seems intuitively reasonable. What we have essentially done is to increase the correlation between groups of covariates that have non-zero coefficients and groups that do not have non-zero coefficients. It thus seems that all methods erroneously select some of the correlated covariates.

In terms of AUC elastic net is still performing the best on average with an AUC of 0.697, though the differences are small compared to ridge and lasso. The SCAD and adaptive lasso perform worse in this respect. Performance measured by Brier score is also similar, but lasso again achieves the lowest score.

6.4.5 Simulation 5: change β coefficients

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.710	0.300
Ridge	1000	0	1	0.1	0.871	0.147
Lasso	50.6 (17.7)	0.968	0.214	0.424	0.899	0.131
SCAD	55.9 (20.51)	0.958	0.182	0.326	0.896	0.131
Elnet (0.92)	51.9 (18.1)	0.967	0.224	0.432	0.899	0.131
Alasso	35.5 (39.8)	0.977	0.145	0.408	0.896	0.131

Table 6.5: Simulation results for increased $\boldsymbol{\beta}$ coefficients (simulation 5), with $m = 200$ runs for each method.

Changing the sizes of the $\boldsymbol{\beta}$ coefficients has a large impact on the results presented in Table 6.5 compared to those given in Section 6.4.4. First, we see that the number of variables selected has been reduced for all models. The lasso on average chooses 50.6 coefficients to be non-zero. This can be explained by the increased sizes of some β-coefficients, and decreasing the sizes of some of the β-coefficients. These now give a weaker signal than before, and the lasso thus does not select them. The SCAD is not as sensitive to these changes as the lasso in terms of number of variables selected, but the accuracy of the selected covariates has been reduced, both in terms of $P_{\hat{\beta}=0}$ and $P_{\hat{\beta}\neq 0}$. The elastic net now selects on average 51.9 coefficients. This is due to the average α value which is now 0.92, quite close to a lasso solution. Adaptive lasso is now the most conservative of the methods, with only 35.5 variables selected on average. Note that it is still affected by a large degree of variance in the number of covariates selected when compared to any of the other methods.

Lasso and elastic net perform similarly in terms of AUC, which is not surprising given that they are indeed very similar as shown by the large average value for α in the elastic net. SCAD and adaptive lasso perform somewhat worse. The most notable difference is that ridge now performs the worst of all methods. Again, this is perhaps not surprising, seeing that the ridge generally tends to dominate over the lasso in situations with many small coefficients, the opposite of the current situation. All the regularization methods perform similarly in terms of Brier score except ridge which performs worst among the regularization methods.

6.4.6 Simulation 6: increase group size to 20

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.709	0.300
Ridge	1000	0	1	0.1	0.874	0.145
Lasso	57.4 (15.8)	0.958	0.192	0.335	0.898	0.130
SCAD	51.6 (15.1)	0.961	0.165	0.319	0.897	0.130
Elnet (0.95)	57.9 (16.0)	0.957	0.191	0.330	0.899	0.130
Alasso	42.0 (60.1)	0.969	0.140	0.333	0.895	0.132

Table 6.6: Simulation results for increased group sizes (simulation 6), with $m = 200$ runs for each method.

All methods with the exception of SCAD select a larger number of covariates when the group sizes increases. In combination with this, $P_{\hat{\beta}\neq 0}$ decreases, which leads to a lower ACC_β. This implies that variable selection has become more difficult, which one may expect due to the increased group sizes. Despite the apparent increased difficulty of variable selection though, all methods perform similarly in terms of prediction (both AUC and Brier score) when compared to Section 6.4.5. While the correlation between all covariates has generally increased, the correlation between the true non-zero variables is the same as in the previous simulation experiment. This is the reason why we were so detailed in our definition of ω in Section 6.1.2. So it appears variable selection has become more difficult in this section, but that the predictive performance does not suffer from this.

6.4.7 Simulation 7: increase n

Increasing the number of observations n leads to an increase in the number of variables selected for all methods. Additionally, all models with the exception of adaptive lasso have become better in terms of ACC_β. Note also that the SCAD and adaptive lasso methods do not benefit more than any of the other methods from the increased sample size, despite their oracle properties. In terms of prediction, there are some slight improvements for all methods in terms of both AUC and Brier score. Most notably, and quite interestingly, the ML fit now performs comparably with the fitted models obtained via regularization. This can be explained by the bias-variance trade-off in Chapter 2. As the number of observations n increases, the variance of the ML estimator $\hat{\boldsymbol{\beta}}$ decreases. There is therefore not much to

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.894	0.132
Ridge	1000	0	1	0.1	0.895	0.131
Lasso	119.1 (18.8)	0.918	0.452	0.379	0.902	0.127
SCAD	65.7 (8.3)	0.961	0.311	0.473	0.901	0.127
Elnet (0.98)	120.4 (18.0)	0.916	0.449	0.374	0.902	0.127
Alasso	112.9 (82.8)	0.913	0.346	0.306	0.901	0.127

Table 6.7: Simulation results for increased number of observations (simulation 7), with $m = 200$ runs for each method.

gain in trying to reduce variance by introducing bias, which is the purpose of these regularization methods.

6.5 Summary

We have now studied five regularization techniques for a wide range of different dependence structures among the covariates. Though the results were quite different for the different simulation experiments, there are some conclusions to be drawn. Lasso and SCAD seemed to be the most conservative methods in terms of number of covariates selected. Due to this, they usually performed best in terms of accuracy on the selected covariates as measured by ACC_β. However, one may argue that the solutions offered by these methods are not as informative exactly because they are so conservative in their selection. The elastic net offered different solutions, and usually selecting a much larger number of covariates. This lead to the inclusion of up to 80% of all non-zero coefficients in the true model, though it also made the elastic net perform worse than lasso in terms of ACC_β. However, one downside to the elastic net is the high variability in the number of covariates selected, most likely caused by the flexibility in the α parameter. Adaptive lasso performed in between lasso and elastic net in terms of variable selection when considering both $P_{\hat{\beta}\neq 0}$ and ACC_β. However it did perform unsatisfactorily when the $\boldsymbol{\beta}$ coefficients were increased in absolute value, in that it selected a very low number of covariates. Even though it was more conservative than the lasso in Sections 6.1.5 and 6.1.6, it got a lower ACC_β than did lasso. Additionally, adaptive lasso was affected by a large degree of variability in the number of variables selected throughout all the different simulation experiments. Admittedly, part of this can be explained by the

lack of optimization with respect to the γ parameter. Because we set $\gamma = 1$, we force adaptive lasso to emphasize the weights generated by the ridge estimator to a degree which may not be optimal. Another possible cause of this uncertainty is the fact that the cross-validation procedure for adaptive lasso had an additional element of randomness compared to for instance the cross-validation procedure for lasso. This can be because one has to compute the initial weights $\hat{w}_j$ from a ridge fit for each fold. The weights used in the different folds are thus all different. Moreover, when we train the final model based on the optimal λ value from cross-validation, we compute yet another set of weights as a part of training the final model. The large variability in the variable selection coupled with the computationally intensive fitting procedure makes the adaptive lasso an unattractive method for our purposes. The ML solution as well as ridge have not been commented much because they do not perform variable selection.

Predictive performance was measured by AUC and Brier score. Lasso performed best in AUC for those scenarios where the $\boldsymbol{\beta}$ coefficients were not all the same. For the other scenarios lasso performed slightly worse than the best method. The lasso often performed best in terms of Brier score as well. Elastic net performed best in terms of AUC for many of the simulation experiments, though the differences between elastic net, ridge and lasso were sometimes small. It seems that the elastic net performed best when lasso and ridge performed similarly. When the $\boldsymbol{\beta}$ coefficients were changed in Section 6.4.5, the elastic net performed similarly to lasso. Adaptive lasso got the lowest AUC of all the regularized models for the scenarios when the $\boldsymbol{\beta}$ coefficients were small. When these were changed to include some large true coefficients, adaptive lasso performed similarly to lasso and elastic net. The SCAD also scored among the lowest in terms of AUC in several experiments, and did not seem to be better than for instance elastic net or lasso for prediction.

Our simulations thus show that lasso and elastic net perform well in terms of variable selection across a wide range of dependence structures among the covariates. Elastic net selects many more variables than lasso which, depending on the situation, one can either consider a pro or a con of elastic net. These two methods are also relatively easy to implement in R, and will fit quite fast compared to for instance adaptive lasso. They were also among the best performers when considering prediction performance as measured by AUC and Brier score.

6.6 A comment on the effects of increased collinearity

In our simulations we observed that as the correlation among the non-zero covariates increased, the AUC increased as well. We will perform some additional small-scale simulations in this section in order to further understand this behavior. Using the same notation, we now consider a scenario where $p_g = 10$, but $n_g = 2$, so there are only 20 covariates. The first group has a constant group correlation, but the latter group is an independence group. We transform all marginals to standard normal variables. Further, $\boldsymbol{\beta}$ is defined such that the first 10 elements are 0.4, and the last 10 are 0, and $\beta_0 = 0$.

It is not obvious why increasing the correlation between the non-zero coefficients should make for an easier classification problem is terms of AUC. To provide an explanation we begin by proposing an expression for a sort of "signal-to-noise ratio" for logistic regression. As a starting point, consider for a moment the setup in linear regression where one assumes the model

$$Y = \underbrace{\mathbf{x}^T\boldsymbol{\beta}}_{\text{signal}} + \underbrace{\varepsilon}_{\text{noise}}.$$

When designing simulation experiments for linear regression models, it is common practice to keep track of the signal-to-noise ratio SNR. That is, a ratio of the variance in the signal to the variance from the noise given by

$$\text{SNR} = \frac{\text{Var}\left(\mathbf{x}^T\boldsymbol{\beta}\right)}{\text{Var}\left(\varepsilon\right)}.$$

The problem in our scenario is that we are working with the logistic regression model

$$\eta(\mathbf{x}) = \underbrace{\mathbf{x}^T\boldsymbol{\beta}}_{\text{signal}}, \quad \underbrace{Y \sim \text{Bernoulli}\left(\exp\{\eta(\mathbf{x})\}/(1+\exp\{\eta(\mathbf{x})\})\right)}_{\text{source of noise}},$$

so the only noise in our simulations comes from the assumption of a binomial distribution for Y. If we wish to find an appropriate definition of signal-to-noise ratio for logistic regression, we must separate the variance into two parts: noise introduced by assumption of Bernoulli distributed Y, and variance in Y actually caused by a stochastic $\mathbf{x}$. To this end, consider the rule of double variance

$$\text{Var}\left(Y\right) = \text{E}\left[\text{Var}\left(Y|\mathbf{x}\right)\right] + \text{Var}\left(\text{E}\left[Y|\mathbf{x}\right]\right). \tag{6.1}$$

Returning to linear regression, and assuming $\mathbf{x}$ and ε are independent of each other, we have

$$\begin{aligned}\mathrm{E}\left[\mathrm{Var}\left(Y|\mathbf{x}\right)\right] &= \mathrm{E}\left[\mathrm{Var}\left(\varepsilon\right)\right] = \mathrm{Var}\left(\varepsilon\right)\\ \mathrm{Var}\left(\mathrm{E}\left[Y|\mathbf{x}\right]\right) &= \mathrm{Var}\left(\mathbf{x}^T\boldsymbol{\beta}\right).\end{aligned}$$

And thus for linear regression, we may write the signal-to-noise ratio in a more general way as

$$\mathrm{SNR} = \frac{\mathrm{Var}\left(\mathrm{E}\left[Y|\mathbf{x}\right]\right)}{\mathrm{E}\left[\mathrm{Var}\left(Y|\mathbf{x}\right)\right]}. \tag{6.2}$$

This formulation of the SNR has the advantage that it can be computed also for logistic regression. Evaluating (6.2) is not always easy, but is made easier by assuming all marginals are standard normally distributed due to the one-to-one relationship between the correlation between the uniforms and the marginals in this situation. Note that since all covariates are normally distributed with mean 0, the linear predictor $\eta(\mathbf{x})$ is also normally distributed with mean 0, and the variance $\sigma^2_{\eta(\mathbf{x})}$ is given by

$$\mathrm{Var}\left(\eta(\mathbf{x})\right) = \sum_{i=1}^{p}\sum_{j=1}^{p}\mathrm{Cov}(\mathrm{x}_i,\mathrm{x}_j)\beta_i\beta_j. \tag{6.3}$$

Using this, we can express the expected conditional variance as

$$\begin{aligned}\mathrm{E}_{\eta(\mathbf{x})}\left[\mathrm{Var}\left(Y|\mathbf{x}\right)\right] &= \int_{-\infty}^{\infty} P(Y=1|\mathbf{x})\left(1-P(Y=1|\mathbf{x})\right) f_{\eta(\mathbf{x})}\left(\eta(\mathbf{x})\right) d\eta(\mathbf{x})\\ &= \int_{-\infty}^{\infty} \frac{e^{\eta(\mathbf{x})}}{1+e^{\eta(\mathbf{x})}}\frac{1}{1+e^{\eta(\mathbf{x})}}\frac{1}{\sqrt{2\pi\sigma^2_{\eta(\mathbf{x})}}}e^{-\eta(\mathbf{x})^2/\left(2\sigma^2_{\eta(\mathbf{x})}\right)} d\eta(\mathbf{x})\\ &= \int_{-\infty}^{\infty} \frac{e^{\eta(\mathbf{x})}}{\left(1+e^{\eta(\mathbf{x})}\right)^2}\frac{1}{\sqrt{2\pi\sigma^2_{\eta(\mathbf{x})}}}e^{-\eta(\mathbf{x})^2/\left(2\sigma^2_{\eta(\mathbf{x})}\right)} d\eta(\mathbf{x})\\ &= \mathrm{E}_{\eta(\mathbf{x})}\left[\frac{e^{\eta(\mathbf{x})}}{\left(1+e^{\eta(\mathbf{x})}\right)^2}\right].\end{aligned} \tag{6.4}$$

Similarly, the variance of the conditional expectation is

$$\mathrm{Var}_{\eta(\mathbf{x})}\left(\mathrm{E}\left[Y|\mathbf{x}\right]\right) = \mathrm{E}_{\eta(\mathbf{x})}\left[\left(\frac{e^{\eta(\mathbf{x})}}{1+e^{\eta(\mathbf{x})}}\right)^2\right] - \mathrm{E}_{\eta(\mathbf{x})}\left[\frac{e^{\eta(\mathbf{x})}}{1+e^{\eta(\mathbf{x})}}\right]^2. \tag{6.5}$$

Such integrals are generally difficult to evaluate analytically, but seeing that these can be written as expectations, we may compute them by Monte Carlo

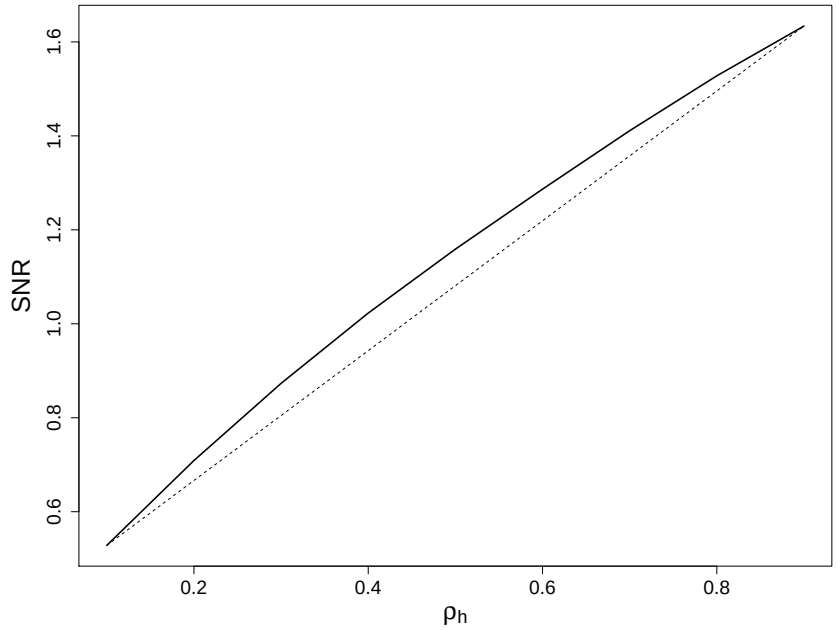


Figure 6.4: SNR plotted against ρ_h. Straight dashed line added for comparison.

methods.

We now run this smaller simulation experiment for values of ρ_h ranging from 0.1 to 0.9, and run $m = 400$ simulations for each of these values. We generate $n = 1000$ observations, of which half are used for training and half are used for testing. We only consider the ML logistic regression model due to the low number of covariates. The signal-to-noise ratio (6.2) is calculated by computing the sum (6.3) directly, and estimating the expectations (6.4) and (6.5) is done by 10^6 Monte Carlo simulations to keep the error low. Additionally, we perform antithetic sampling which further reduces error (Bølviken, 2014). Figure 6.4 shows the relationship between ρ_h and SNR for the logistic regression model defined in this section. From this plot we can see that increasing ρ_h increases the signal-to-noise ratio. Figure 6.5 shows AUC as a function of ρ_h. In light of the insight provided by Figure 6.4 it comes as no surprise that increasing ρ_h leads to an increase in AUC, since we are essentially just increasing the SNR. Thus, increasing correlation among covariates can be seen as a way of adjusting the SNR in logistic regression. In terms of AUC, increasing ρ_h is thus similar to increasing the sizes of the $\boldsymbol{\beta}$ coefficients.

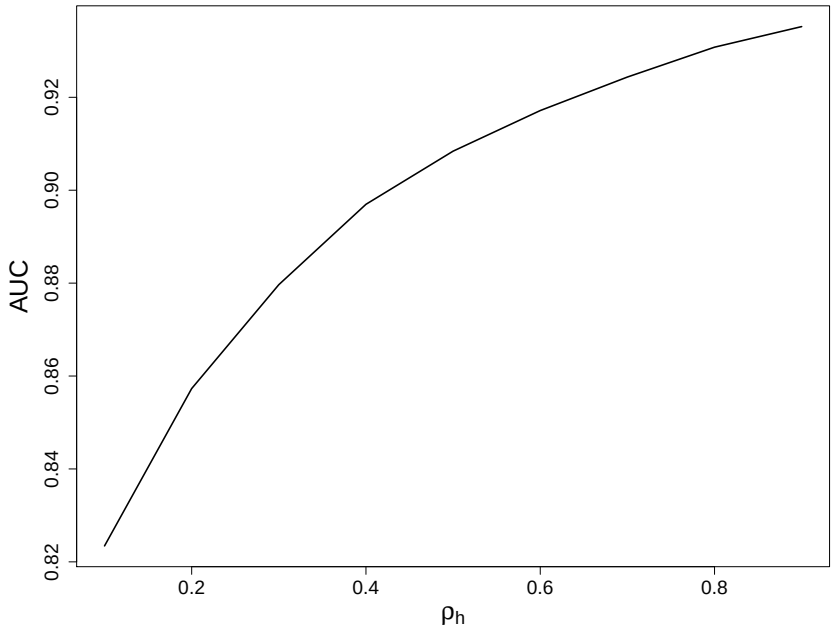


Figure 6.5: AUC plotted against ρ_h.

Chapter 7

Simulation study: class imbalance

In this chapter we will conduct simulation experiments to study whether class imbalance can cause issues when training logistic regression models. There are many studies pointing to the problems of unbalanced training data and possible remedies for this (Solberg and Solberg, 1996), (Japkowicz, 2000). Common for many of these studies is that they focus mostly on tree-based methods such as the algorithm C4.5 (Chawla et al., 2004). Since our focus is on logistic regression, many of the papers where only tree-based methods are considered are not of much interest. There are however some studies discussing the effects of class re-sampling on a logistic regression model. For instance in Van Hulse et al. (2007) one compared the AUC of a logistic regression model trained both on original unbalanced data and a re-sampled balanced training set. Improvements in AUC were modest, and the AUC obtained by no sampling compared to random oversampling were different only in the 3rd decimal. Statistical tests were also set up to test whether the improvement of random oversampling to the AUC was significant. The conclusion was that the random oversampling technique, which performed best for logistic regression in terms of AUC, did not lead to a significant improvement. In Oommen et al. (2011) one found that class imbalance as the result of bias in the class balances lead to biased prediction of probabilities, and that sampling techniques could help in reducing the bias by restoring the original class balances. However, it was also here concluded that sampling techniques do not improve the AUC of a logistic regression model. It should be noted that the construction of the simulation experiments in Oommen et al. (2011) did not allow for a thorough study of the effects of class imbalance on AUC.

An aspect of the class imbalance problem that to our knowledge has not been addressed in the literature is whether the combination of a high number of covariates and class imbalance can have an unfortunate effect on variable selection. In most studies the number of covariates is kept low, with the largest number of covariates studied being 65 in Van Hulse et al. (2007). A comparison of regularization techniques in the presence of class imbalance has to our knowledge not been done either. Lastly, looking at the problem of class imbalance and variable selection separately is not realistic, and studying the combined effect of these two problems is very much of interest. In conclusion, it seems that logistic regression does not benefit from re-sampling techniques in terms of AUC, but the overall picture is not clear. Further, re-sampling methods have not been applied to data of particularly high dimension, and the effects of re-sampling methods on variable selection have not been discussed. We hope to provide some clarity on these issues in this chapter.

We will use the model defined in Section 6.1.6 as the basis for all simulations in this chapter. The class balance will be measured by $\mathrm{E}[Y]$ when referring to the properties of a stochastic model from which we sample data, or $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$ when referring to a particular data set.

7.1 Adjusting class balance

Acquiring a desired class balance can be done in several ways which we will now discuss. One way is to generate a large data set from a model where $\mathrm{E}[Y] = 0.5$, and then sample without replacement a given number of positive and negative samples such that $\bar{Y}$ reaches a desired level. Alternatively, we could generate data from a model where $\mathrm{E}[Y]$ is already at the desired level to begin with. How one does this is not obvious, and many different approaches can be taken. The different data generation strategies also have different interpretations. In the former, we assume that fraudulent behavior remains constant, but that our training data are biased. In the latter, we adjust what defines fraudulent behavior, and assume our data are unbiased. These two strategies for generating unbalanced data are presented in Oommen et al. (2011). However, their simulations did not allow for a thorough study of the AUC for these different data generation strategies, so it is therefore difficult to know beforehand which data generation strategy would make for a most interesting simulation study. We have chosen to generate data using the second approach. This is because we can not state with any certainty that sampling bias is indeed a problem for our VAT data.

We therefore instead choose to reduce the class balance by increasing the threshold of what defines fraud. To reduce the class balance we will reduce the value of β_0. The interpretation of this is that as β_0 is reduced, we require fraudulent behavior to be more extreme, in the sense that the covariates must be of greater magnitude to counter the large negative β_0.

Deciding the value of β_0 to obtain a desired class balance other than $\mathrm{E}[Y] = 0.5$ requires some work. Recall that we argued that if $\mathrm{E}\left[\eta(\mathbf{x})\right] = 0$ then our class balance would be $\mathrm{E}\left[Y\right] = 0.5$. However, if we desire a different class balance, say $\mathrm{E}\left[Y\right] = 0.2$, we need not only take into consideration what the value of $\mathrm{E}\left[\eta(\mathbf{x})\right]$ should be, but also $\mathrm{Var}\left(\eta(\mathbf{x})\right)$. The formula for $\mathrm{Var}\left(\eta(\mathbf{x})\right)$ in (6.3) is still valid, but the covariances are not known in the case of non-normal marginal distributions, as noted in Chapter 5. Deducing the correct β_0 value for a desired level of class balance is therefore difficult analytically. We therefore again turn to Monte Carlo methods to solve the problem. This can be done by simulating m observations from the stochastic model defined in Section 6.1.6 for k different values of β_0. The class balance is then estimated by $\frac{1}{m}\sum_{i=1}^{m} Y_i$ in each case. This approach is costly is terms of execution times if we want a fine grid of β_0 values since we in total will have to generate mk data points from the model in Section 6.1.6. We can however make the problem a little easier. First, we know from Section 6.1.6 that the expected class balance is exactly 0.5 when $\beta_0 = -3.4$, thus we need not estimate the class balance for larger values than this. We will use linear interpolation to estimate the class balance for β_0 values between those k estimated directly. The resulting graph is given in Figure 7.1. Each dot represents one of the k estimates, and the line between the dots is obtained by linear interpolation. Linear interpolation is perhaps somewhat simplistic, but seems to do a decent job in this case, since $\mathrm{E}[Y]$ does not seem to be too non-linear for such an approximation to be inappropriate. The choice of β_0 can then be read off Figure 7.1 based on the desired value of $\mathrm{E}[Y]$. We will in the most unbalanced situation train the logistic regression model on data sets with $\mathrm{E}[Y] = 0.01$. What then often happens is that we generate data sets where there are no observations from the positive class, or too few to train a model. For instance, the cross validation procedure in the `glmnet` package issues a warning if there are fewer than 9 observations of each class in the training set. To overcome this problem, we will begin each of our simulations by generating one large data set of size $5n$ to ensure at least with a reasonable large probability that there are enough observations of both classes to make a data set of size n. We then sample without replacement from this data set such that we can reach exactly the desired class balance for each simulation. Again, this strategy for generating unbalanced data

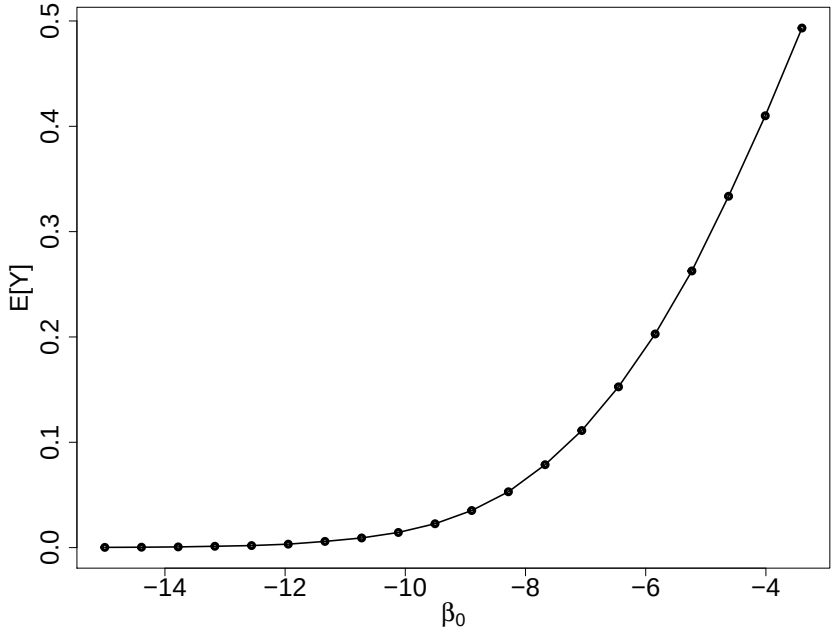


Figure 7.1: E[Y] plotted as a function of β_0.

sets is outlined in Oommen et al. (2011).

7.2 Experiment design

The following simulation experiments are set up as in Section 6.1.6, and thus all parameter values are given there unless otherwise specified.

7.2.1 Simulation 1: $\mathbf{E}[\boldsymbol{Y}] = 0.2$

A natural starting point is to generate data where the class balance equals that of the VAT data set, in which $\bar{Y} \approx 0.2$. We set $\beta_0 = -7.7$ to achieve a class balance of $\mathrm{E}[Y] = 0.2$.

7.2.2 Simulation 2: $\mathbf{E}[\boldsymbol{Y}] = 0.05$

We keep the setup as in Simulation 1, but with a reduction of the class balance to $\mathrm{E}[Y] = 0.05$ by setting $\beta_0 = -10.3$.

7.2.3 Simulation 3: $\mathbf{E}[\boldsymbol{Y}] = 0.01$

Lastly, we further reduce the class balance to $\mathrm{E}[Y] = 0.01$ by setting $\beta_0 = -12.5$.

7.2.4 Simulation 4: $\mathbf{E}[\boldsymbol{Y}] = 0.01$, re-sample to $\bar{\boldsymbol{Y}} = 0.2$

We sample data from the model defined in Simulation 3, but perform re-sampling on the training data so that $\bar{Y} = 0.2$. The re-sampling methods used are those given in Chapter 4, namely under-sampling, random over-sampling and SMOTE.

7.2.5 Simulation 5: $\mathbf{E}[\boldsymbol{Y}] = 0.01$, re-sample to $\bar{\boldsymbol{Y}} = 0.5$

The same underlying true model as in Simulation 3, but we re-sample the training data such that $\bar{Y} = 0.5$ using the same re-sampling methods as used in Simulation 4. Thus, we simply perform more re-sampling compared to Simulation 4.

7.3 Implementation and estimation

The setup of the simulation experiments also builds on that given in Chapter 6. Running the simulations will be done as described by the pseudocode given in Algorithm 6.1.

Cross validation

Some comments on how one implements a re-sampling scheme into a cross-validation procedure is worth mentioning. We will now discuss the potential pitfalls when combining re-sampling with cross validation.

The first approach is to perform re-sampling on the training set prior to the cross-validation procedure, and then run cross validation as one normally would. This can be harmful to the training process when performing both under-sampling and over-sampling. We begin by taking a closer look at the former. When under-sampling is performed on the training set, we are essentially throwing away observations belonging to the negative class (the 0's). This is unfortunate in itself, since these data could have been used for

testing in the cross-validation procedure. However, what is potentially more troubling is that each test set in the cross-validation procedure will now be balanced, and thus does not resemble the final test set in this respect. This can be problematic when the measure of predictive performance used is sensitive to the class balance. Seeing that we have been careful not to re-sample the true test set, we should not be satisfied by re-sampling the test sets in the cross validation procedure either. When over-sampling is performed the effects of re-sampling prior to the cross-validation procedure can be even more harmful. Consider first random over-sampling where we inflate the minority class by adding copies of the original observations. First note that we have the same problem as that explained above, since the balance of the test set in the cross validation procedure does not resemble that of the final test set. A more troubling issue emerges as a result of over-sampling, namely that one observation, say $(Y^*, \boldsymbol{X}^*)$ may be re-sampled several times, and appear in more than one fold in the cross validation procedure. We have thus violated one key property of cross validation, since the folds are no longer disjoint data sets. One observation may then very well appear in several, or all folds. This introduces bias in the estimation of model performance, and will lead to over-fitting.

Since data re-sampling prior to cross validation appears to be inappropriate, we consider alternatives. A naturally occurring alternative is to re-sample each training set inside the cross-validation procedure. The problem with a mismatching class balance of the test sets in cross validation and the final test set has been solved. In addition, over-sampling in this case will not cause over-fitting, since we keep the training and test sets separate for each iteration in the cross-validation procedure, and only over-sample based on the current training set. However, there appears to be one problem with this approach as well. Re-sampling the training set separately in each iteration of cross-validation will lead to an unnecessary randomness in the data, since the training set can vary quite substantially for each iteration. Further, when the cross-validation procedure has finished, and we wish to train the final model based on the optimal hyper-parameters, we will re-sample the training set once more. The training sets in the cross validation procedure are thus riddled with potentially unnecessary variance, in addition to the final training set being different from those training sets used in cross validation. It should be mentioned that during our work on this thesis a paper by Santos et al. (2018) was published. The combination of cross-validation and re-sampling described here is similar to their implementation, which was not discovered until after the simulations in this chapter were run. However, as noted above, there may be potential

issues with this approach, so we instead opted for a different combination of cross-validation and re-sampling which we will now describe.

We propose an alternative approach for combining re-sampling and cross-validation. We begin by applying Algorithm 2.1 on the training data. For each fold, we then re-sample data contained in only that fold, and label these re-sampled data. For under-sampling this means that those negative observations which should be removed are labeled. Conversely, for over-sampling the new, synthetic observations are labeled. In the case of under-sampling, the labeled observations are included in the test-set, but not the training set in each iteration of the cross-validation procedure. For over-sampling we include the labeled observations in the training set, but not the test set for each iteration in the cross-validation procedure. This ensures that the test sets in the cross validation procedure are of the correct balance compared to the final test set. It also means that each fold contributes with exactly the same data in each iteration of the cross validation procedure, and further that the final model can be trained on exactly the same data as was used in cross validation.

Lasso and elastic net

We will use lasso and elastic net in the simulation experiments of this chapter. This is because we observed that these two methods performed best both in terms of prediciton, but also for variable selection in Chapter 6.

Random forest

As noted earlier, most of the literature on unbalanced data sets focus on tree based methods. In order to keep the following simulation experiments in touch with results in other research, we will use a random forest method as well as (penalized) logistic regression. For this reason we briefly introduce the concept of random forests in this section. Random forests (Breiman, 2001) are an extension of bagging (Breiman, 1996a) in the case of regression/classification trees. In bagging one samples the original training set $\mathbf{Z} = (\mathbf{X}, \mathbf{Y})$ with replacement to construct a new training set $\mathbf{Z}' = (\mathbf{X}', \mathbf{Y}')$. This new training set is then used to train a given model $f(\mathbf{x})$. This procedure is repeated B times, as shown in Algorithm 7.1. Final predictions of a bagged model is then made by averaging the predictions of all B models, i.e. $f_{bag}(\mathbf{x}) = \frac{1}{B}\sum_{i=1}^{B} f_i(\mathbf{x})$. The models $f_i(\mathbf{x})$ for $i = 1, \ldots, B$ can in principle be any kind of model, but trees are commonly used. This has to do with the large variance of trees. When trees are bagged, one can reduce the variance

Algorithm 7.1 Pseudocode for bagging.

1: **For** i in 1:B **do**
2: Create bootstrap sample $\mathbf{Z}'_i$ from $\mathbf{Z} = (\mathbf{X}, \mathbf{Y})$
3: Train $f_i(\mathbf{x})$ on $\mathbf{Z}'_i$
4: **End For**

of predictions by possibly quite a large extent. Assuming that the variance of all models is given by $\text{Var}\left(f_i(\mathbf{x})\right) = \sigma^2$ for $i = 1, \ldots, B$, a rationale for bagging can be seen by

$$\begin{aligned}
\text{Var}\left(f_{bag}(\mathbf{x})\right) &= \frac{1}{B^2}\text{Var}\left(\sum_{i=1}^{B} f_i(\mathbf{x})\right) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B}\text{Cov}\left(f_i(\mathbf{x}), f_j(\mathbf{x})\right) \\
&= \sigma^2\frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B}\text{Cor}\left(f_i(\mathbf{x}), f_j(\mathbf{x})\right),
\end{aligned}$$

such that if $\text{Cor}\left(f_i(\mathbf{x}), f_j(\mathbf{x})\right) < 1$ for any pair $i \neq j$ the predictions of our bagged model will have smaller variance. The goal is then to produce models $f_i(\mathbf{x}), i = 1, \ldots, B$ which have as low correlation as possible. This can be a problem especially in situations where there are a only few covariates with a strong signal. Using trees in combination with bagging may then result in great similarity in the splits of the trees. As a result, the predictions made by such trees will be highly correlated which negatively affects the predictive performance of the final bagged model. This is where the random forest approach comes in. Random forest keeps the same framework as that used for bagging, but with one key difference having to do with the construction of the trees. The tree-growing algorithm is only allowed to consider $m < p$ covariates at each split, forcing a greater exploration of the predictor space. Random forests have proven to be versatile and easy to implement methods for classification. While one can certainly benefit from tuning m and/or B using for instance cross-validation, it has been observed that random forests perform very well with little parameter tuning. This is in fact one of its strengths. For the following simulations we set the number of trees $B = 500$ and the number of covariates considered at each split to $\sqrt{p}$ which has been recommended for classification in Hastie et al. (2009). We will use the package `randomForest` to fit random forest models in R.

SMOTE

We will set $p = 5$ in the SMOTE algorithm as recommended by Chawla et al. (2002). In the SMOTE algorithm there is an additional parameter m which decides how many synthetic observations should be generated for each positive observation. In these simulation experiments we round m up to the smallest integer such that we generate enough synthetic samples to obtain the desired class balance. We then re-sample without replacement from these synthetic observations until we reach the desired balance.

7.4 Results

The model evaluation criteria presented in Section 6.3 will be used also in these simulation studies. The measures of predictive performance are computed on a test set of size $n = 3000$. The class balance $\mathrm{E}[Y]$ of the test set is the same as that for the training set, and no re-sampling is applied to the test set.

7.4.1 Simulation 1: $\mathbf{E}[Y] = 0.2$

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.709	0.238
Lasso	50.7 (14.3)	0.963	0.171	0.337	0.909	0.091
Elnet (0.95)	52.3 (17.5)	0.961	0.174	0.332	0.909	0.091
RF	-	-	-	-	0.863	0.112

Table 7.1: Results for running Simulation 1 $m = 200$ times for each method.

In this simulation we set the class balance at $\mathrm{E}[Y] = 0.2$ as opposed to $\mathrm{E}[Y] = 0.5$ which was the case in Section 6.4.6. The results are shown in Table 7.1, and when compared to the results in Table 6.6 we can see some differences. The average number of variables selected $N_{\hat{\beta}\neq 0}$ has decreased for both lasso and elastic net. The α parameter for elastic net averages at 0.95, similar to the average α value in Table 6.6. As a result of the low number of variables selected, fewer of the true non-zero covariates are selected, as we see by a reduction of $P_{\hat{\beta}\neq 0}$. It seems that lowering class balance has slight detrimental effects on variable selection.

The AUC has increased when compared to the figures shown in Table 6.6. This should not be taken to mean that the fitted models have become better in terms of prediction. It seems generally that as we decrease the intercept β_0 in the true model the AUC will increase. The same behavior can be seen in the results of Simulation 2. Random forest obtains an AUC of 0.863 which is lower than the penalized logistic regression fits, but better than maximum likelihood logistic regression. The fact that random forest performs worse than penalized logistic regression should come as no surprise, since we are indeed generating data from a logistic regression model. Brier score has reduced from approximately 0.13 for all methods in Table 6.6 to 0.09 for lasso and elastic net in this scenario. Random forest scores worse than the penalized LR models, but better than maximum likelihood LR in terms of Brier score. In conclusion, it seems that the methods for fitting the logistic regression model at least do not suffer in terms of AUC when the class balance is $\mathrm{E}[Y] = 0.2$.

7.4.2 Simulation 2: $\mathbf{E}[\boldsymbol{Y}] = 0.05$

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.685	0.109
Lasso	43.8 (19.8)	0.966	0.134	0.307	0.924	0.034
Elnet (0.88)	45.2 (19.6)	0.965	0.133	0.296	0.924	0.034
RF	-	-	-	-	0.840	0.040

Table 7.2: Results for running Simulation 2 $m = 200$ times for each method.

We have now reduced the class balance to 0.05. The number of variables selected has decreased further from the results given in Table 7.1. In addition to this, the accuracy of the selected covariates has also been reduced, as can be seen by a reduction in ACC_β for both lasso and elastic net. Thus, the penalized logistic regression fits select both fewer covariates, and loses accuracy on those covariates that are selected. The average α value for elastic net has been reduced to 0.88.

The maximum likelihood logistic regression model performs worse with respect to AUC when compared to results in Table 7.1. However the penalized fits perform similarly, even increasing in AUC. Again, this increase is due to the further reduction of β_0 compared to Simulation 1. It does

not seem that the penalized logistic regression models suffer in terms of prediction. Random forest performs similarly in terms of AUC compared to the results given in Table 7.1, though somewhat worse. Note also that all methods seem to improve the Brier score compared to results in Table 7.1.

7.4.3 Simulation 3: $\mathbf{E}[\boldsymbol{Y}] = 0.01$

Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
No	1000	0	1	0.1	0.660	0.014
Lasso	29.8 (21.5)	0.975	0.074	0.248	0.917	0.009
Elnet (0.53)	107.1 (136.2)	0.900	0.175	0.164	0.915	0.009
RF	-	-	-	-	0.586	0.009

Table 7.3: Results for running Simulation 3 $m = 200$ times for each method.

Variable selection seems to have become a greater issue now that the balance has been reduced to $\mathrm{E}[Y] = 0.01$. The number of selected variables has been reduced to 29.8 for lasso, but increased to 107.1 for elastic net. The latter can be explained by the average α value which is now 0.514. Upon closer inspection, we see that the chosen α value are spread between 0.01 and 0.99, so the average of 0.514 is somewhat misleading by itself. Figure 7.2 shows how α is distributed in the 200 simulations. The α values seem to be nearly uniformly distributed, with perhaps a leaning towards $\alpha = 0.99$. This uniformity in the distribution of α is unlike those of the previous simulations in both this chapter and Chapter 6, see for instance Figure 6.3. In the previous simulation studies the distribution of α leaned towards value close to either 0 or 1 as we saw in Figure 6.3. It seems that the low number of 1's in the data induces large variability in α. This in turn introduces high variability in $N_{\hat{\beta}\neq 0}$. The standard deviation of $N_{\hat{\beta}\neq 0}$ for elastic net has increased from 19.6 in Table 7.2 to 136.2. Elastic net succeeds in including more than twice as many of the true non-zero variables in its model than lasso, which comes at the cost of the low accuracy ACC_β.

In terms of prediction for the logistic regression methods we see a decrease in AUC compared to the AUC in Table 7.2, as opposed to an increase which we would expect due to the reduction of β_0. A probable reason for this is that both lasso and elastic net now struggle with variable selection. At some point, this is bound to affect also the predictive ability of a model,

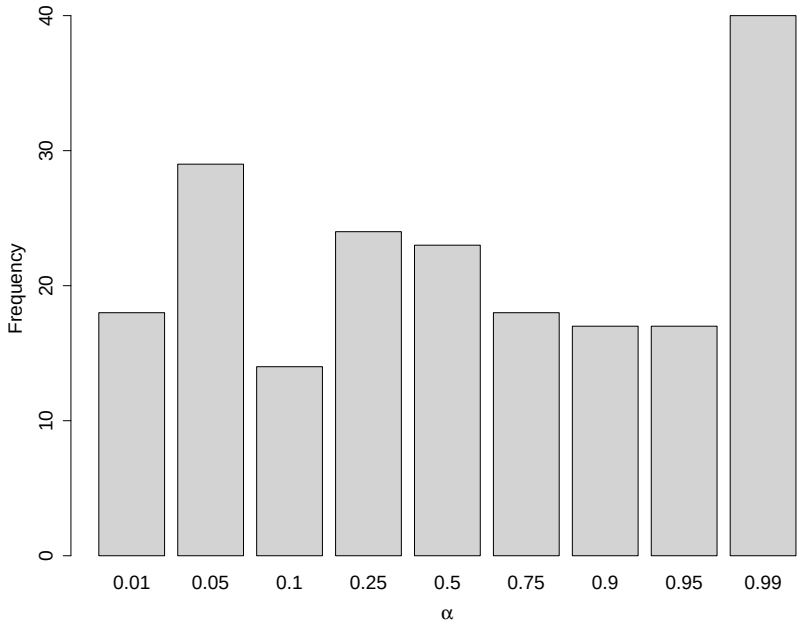


Figure 7.2: Distribution of α for elastic net.

which may well be what we are seeing here. The random forest model seems to completely collapse in terms of AUC. It scores an AUC of 0.586 which indicates it is only slightly better than chance. All methods again achieve a better Brier score compared to that shown in Table 7.2.

7.4.4 Simulation 4: $\mathbf{E}[\boldsymbol{Y}] = 0.01$, re-sample to $\bar{\boldsymbol{Y}} = 0.2$

We have now generated data exactly as in Simulation 3, but performed re-sampling until the class balance is $\bar{Y} = 0.2$. Re-sampling methods are those given in Chapter 4. Results in Table 7.4 should therefore be compared to those in Table 7.3. Under-sampling has not been performed for maximum likelihood logistic regression. The reason for this is that under-sampling to $\bar{Y} = 0.2$ causes $n < p$, in which case the maximum likelihood logistic regression is not appropriate. Under-sampling the majority class seems to have a negative effect on variable selection for both lasso and elastic net. Both methods select fewer covariates when compared to no re-sampling. Elastic selects fewer covariates when random over-sampling and SMOTE is performed. Lasso does not appear to be affected by neither random over-sampling or SMOTE in terms of variable selection.

Re-sampling	Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_{β}	AUC	BS
Under	Lasso	20.4 (13.9)	0.983	0.054	0.264	0.906	0.039
	Elnet (0.44)	74.7 (85.9)	0.932	0.133	0.179	0.907	0.036
	RF	-	-	-	-	0.883	0.042
Over	No	1000	0	1	0.100	0.657	0.016
	Lasso	31.7 (23.0)	0.973	0.076	0.240	0.918	0.027
	Elnet (0.53)	90.7 (117.1)	0.916	0.154	0.170	0.915	0.026
	RF	-	-	-	-	0.714	0.009
SMOTE	No	1000	0	1	0.100	0.655	0.016
	Lasso	30.1 (25.0)	0.974	0.072	0.238	0.912	0.026
	Elnet (0.55)	84.8 (114.7)	0.923	0.151	0.178	0.915	0.026
	RF	-	-	-	-	0.834	0.010

Table 7.4: Results for running Simulation 4 $m = 200$ times for each method.

For under-sampling, the predictive performance has been reduced, with the AUC now 0.906 and 0.908 for lasso and elastic net. Random forest however seems to benefit greatly from under-sampling. Random over-sampling and SMOTE do not appear to impact the AUC for any of the logistic regression models. Random forest behaves differently compared to logistic regression for the over-sampling methods as well, with an increase in AUC for both random over-sampling and SMOTE. It seems that SMOTE offers greater improvements to AUC than does random over-sampling for the random forest model. Focusing on random over-sampling and SMOTE we see that maximum likelihood LR, elastic net and lasso all get a higher Brier score compared to those in Table 7.3, which indicates worse predictive performance. The AUC for these methods on the other hand generally shows that there is very little change in the predictive performance.

7.4.5 Simulation 5: $\mathbf{E}[\boldsymbol{Y}] = 0.01$, re-sample to $\bar{\boldsymbol{Y}} = 0.5$

We still generate data as in Simulation 3, and re-sampling methods are the same as in Simulation 4. The difference is that we now re-sample such that $\bar{Y} = 0.5$. Results in Table 7.5 should therefore be compared to the results given in Table 7.3 and Table 7.4. Further under-sampling of the minority class brings about substantial difficulty in variable selection for lasso and

Re-sampling	Penalty	$N_{\hat{\beta}\neq 0}$	$P_{\hat{\beta}=0}$	$P_{\hat{\beta}\neq 0}$	ACC_β	AUC	BS
Under	Lasso	11.2 (8.7)	0.991	0.031	0.280	0.885	0.164
	Elnet (0.34)	68.7 (93.6)	0.937	0.119	0.173	0.898	0.166
	RF	-	-	-	-	0.892	0.161
Over	No	1000	0	1	0.100	0.657	0.016
	Lasso	30.4 (26.2)	0.975	0.077	0.253	0.915	0.082
	Elnet (0.49)	82.9 (87.2)	0.925	0.154	0.186	0.916	0.088
	RF	-	-	-	-	0.624	0.010
SMOTE	No	1000	0	1	0.100	0.653	0.016
	Lasso	33.4 (32.7)	0.972	0.079	0.237	0.910	0.076
	Elnet (0.54)	76.9 (86.2)	0.930	0.140	0.182	0.914	0.084
	RF	-	-	-	-	0.788	0.010

Table 7.5: Results for running Simulation 5 $m = 200$ times for each method.

elastic net. Fewer variables have been selected which leads to a reduction in $P_{\hat{\beta}\neq 0}$. We saw in Table 7.4 that over-sampling and SMOTE made variable selection more difficult for elastic net, which can be seen here as well. Both the standard deviation and the average number of variables selected for elastic net have been reduced regardless of re-sampling method.

For under-sampling in combination with logistic regression the AUC has been reduced regardless of how the model was fit compared to the results in Table 7.4. Random over-sampling and SMOTE seem to offer no improvement to the logistic regression models in terms of AUC when the degree of re-sampling has been increased. Random forest seems to benefit from increased under-sampling as we can see by the increase in AUC. However, random over-sampling to $\bar{Y} = 0.5$ in combination with random forest leads to a decrease in AUC when compared to random over-sampling only to $\bar{Y} = 0.2$. SMOTE also seems to result in a lower AUC for the random forest when compared to results in Table 7.4.

7.5 Summary

In these simulation experiments we have seen that reducing the class balance by reducing the intercept β_0 in the true data generation model makes variable selection more difficult for the logistic regression model. Depending on the degree of imbalance, the number of variables selected is reduced,

with less accuracy in those variables that are selected. Additionally, we observed that the number of variables selected tends to vary increasingly as the class balance is reduced. This is particularly true for the elastic net, which struggles with large variability in the selection of α when class balance is low. Elastic net thus seems to be an unsuited method when the class balance reaches such low levels. However, the predictive performance of the penalized logistic regression model did not seem to suffer much unless the class balance reaches very low levels at around $\mathrm{E}[Y] = 0.01$. Thus, it seems that logistic regression suffers less from class imbalance than does tree methods such as random forest. To the degree that class imbalance is a problem for logistic regression, this is due to poor variable selection in such situations. This would imply that the class imbalance problem for logistic regression is not caused by the class balance itself, but the low number of positive samples.

Regarding the re-sampling methods, it seems based on these simulation experiments, that none of the re-sampling methods used are able to improve variable selection or predictive performance for a logistic regression model. We can thus conclude that it does not appear that the logistic regression model with a large number of covariates benefits from re-sampling the training set, at least not for any of the re-sampling methods used in this thesis. These results are consistent with studies of re-sampling techniques used for logistic regression in lower dimensions. The results obtained for the random forest model are also consistent with the results found elsewhere in the litterature. We saw that predictive performance of the random forest as measured by AUC can suffer substantially when the class balance reaches low levels, in this case 0.01. Further, in this situation under-sampling seemed to offer the best improvements to the AUC, with SMOTE coming in second, and random over-sampling being the worst of the three.

Chapter 8

Modeling VAT fraud

We have gained some insight into how the different regularization and re-sampling methods behave in the case of dependent and unbalanced data in earlier chapters. In this chapter we will apply the insight gained from the previous chapters to our analysis of the VAT data set.

8.1 Data pre-processing

We began this thesis with an introduction to the VAT data set, but we did not go into much detail. Due to the anonymous nature of the data there is not much that can be said, but we will provide some additional properties of the data set. The data set contains one additional dimension which we have not yet discussed, namely time. For each control, there has been assigned a number between 1 and 39 indicating the time of control. The start and end dates are not known, but time is indexed in terms of size 1/6th of a year. Thus, with the time variable being maximum 39 we have data spanning 6,5 years. This variable will not be used in the predictive model, but will be used when we divide the data into training, validation and test sets. Figure 8.1 shows the number of controls, number of fraudulent cases, and the class balance over time. The number of cases in each term varies around 1500, and the class balance is approximately $\bar{Y} = 0.2$. Upon inspection of Figure 8.1 we notice some anomalies at the first two terms and last term of the data set. For the first two terms we see that the number of controls is very low, and the class balance is much larger than in the rest of the data set. One may then question if the data from these two months are representative of the rest of the data. For this reason we will omit the first two terms in the proceeding analysis. The last term also seems to be quite different in nature compared to the others. The number of cases is again much lower

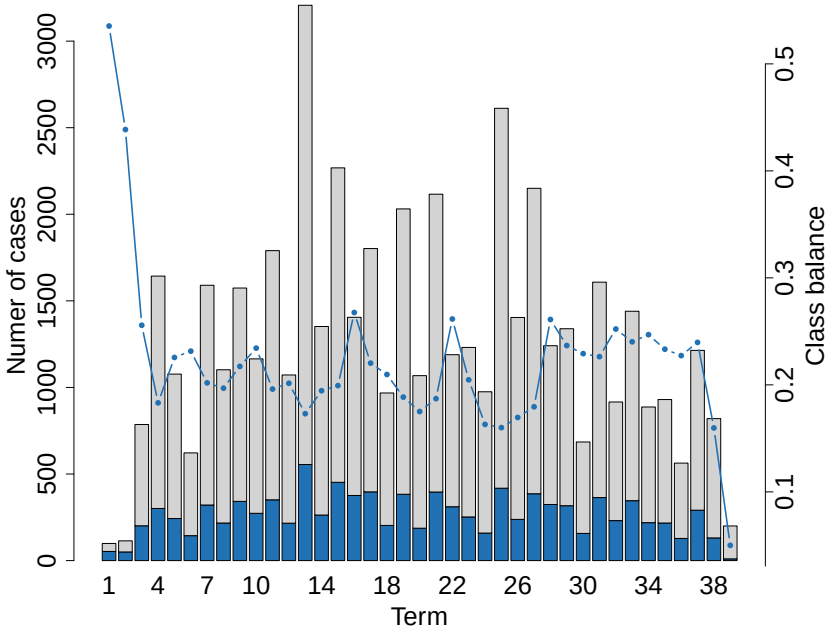


Figure 8.1: Number of controls (grey), number of fraudulent cases (blue) and class balance plotted over time (dark blue line).

than the other terms, and the class balance is substantially lower, so we exclude also the last term from the data set in the following.

One property of the VAT data set which is commonly met in many applications is that of missing, or incomplete data. Data can be missing for several reasons which we will now discuss briefly. Assume we have data $X_i = (X_{i,1}, X_{i,2})$ for $i = 1, \ldots, n$, where for a given j, $X_{j,1}$ is missing but $X_{j,2}$ is observed. The nature of the missing data can be divided into one of three categories. The first category is called *missing completely at random* (MCAR) and is the case when

$$P\left(X_{j,1} \text{ is missing} | X_j\right) = P\left(X_{j,1} \text{ is missing}\right).$$

This means that whether or not $X_{j,1}$ is missing does not depend on the (possibly unobserved) values of either $X_{j,1}$ or $X_{j,2}$. Alternatively, data can be *missing at random* (MAR), meaning that

$$P\left(X_{j,1} \text{ is missing} | X_j\right) = P\left(X_{j,1} \text{ is missing} | X_{j,2}\right).$$

In words, this means that whether $X_{j,1}$ is missing does depend on the value of $X_{j,2}$. The last case is when data is *not missing at random* (NMAR) which

means that the probability of missing data depends on the (unobserved) value of the missing data itself, i.e.

$$P\left(X_{j,1} \text{ is missing}|X_j\right) = P\left(X_{j,1} \text{ is missing}|X_{j,1}\right).$$

Imputation methods are often considered in view of which of these situations one is faced with for a given covariate. This in turn requires information about how the data were collected, and the relationship between the covariates (Hastie et al., 2009). To some degree we do have such information because each covariate has been assigned to one of four possible variable categories. These are: *background*, *current assignment*, *previous terms* and *previous years*. The first category, *background* contains variables that explain general background information about the business being controlled, *Current assignment* contains information relevant for the current control, *previous terms* information from previous cases delivered in previous terms and *previous years* contains information about the business reported during the past years. Thus, for a given case the number of employees at the time of control could be one of the variables in the category *background*, and number of employees the previous years could belong to the category *previous years*. Missing data thus occurs naturally, since for instance newly started businesses will not have any information about previous years or previous terms. However, besides this we have no information about the data collection methods for the VAT data set. For this reason we will not go into details of different data imputation methods, but simply apply median imputation for numerical data. Thus for each missing observation for a specific covariate, we replace the missing value with the median of the observed instances of that covariate. Additionally we will create a new indicator variable which equals 1 if that particular observation was imputed, and 0 else. In the case of missing categorical variables we will create a new category *missing*. If there is information in whether or not a variable is missing, we will thus be able to extract that information and use it in our predictive model. An example of this imputation scheme is given in Table 8.1. These indicator variables can also serve as an indication of whether a given business has handed in reports in previous terms or if information about the business back in time is available. This information could certainly be valuable in an analysis, so there is some justification to the chosen imputation scheme. Doing this increases the number of covariates in our data set. Initially there were 556 covariates of which 539 were numerical and 17 were categorical. After the numerical covariates have been imputed we gain an additional 321 covariates which are indicator variables as described

$\mathbf{x}_1$	$\mathbf{x}_2$		$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_{1,\text{missing}}$
1	Level 1		1	Level 1	0
3	Level 2	$\Rightarrow$	3	Level 2	0
NA	Level 1		2	Level 1	1
2	NA		2	missing	0

Table 8.1: Illustration of imputation scheme.

above. We also convert all categorical covariates into numerical covariates. This is done by converting a categorical covariate with d levels into $d-1$ indicator variables. This gave an additional 85 covariates. One covariate had a standard deviation of 0, and was thus removed. This gives a final count of 944 covariates in the data set.

8.2 Model training

In Chapter 6 we saw that elastic net performed well across many different dependence structures both in terms of prediction and variable selection. However, there were situations in which lasso performed equally well. Ridge also performed well for some of the simulation experiments. For this reason we will use these three methods when modeling VAT fraud. We optimize λ and α by 10-fold cross validation as in the previous chapters. To reduce the variance of our cross-validation estimates of predictive performance, we will run 5 cross validation procedures for each of these methods. We then average the resulting 5 estimates of AUC, and select the parameters that maximize this average. This is known as repeated cross-validation, see Kim (2009). A natural competitor would be maximum likelihood logistic regression. However, it struggles with convergence which results in an AUC lower than 0.5 and will thus not be presented. We will not use any of the re-sampling methods previously discussed in this thesis. This is because it seems, based on the results from Chapter 7 that the logistic regression model does not suffer much from class imbalance when the balance is $\bar{Y} = 0.2$. This is particularly true for prediction as measured by AUC, but also for variable selection. Additionally, as we saw from the simulation experiments conducted in Chapter 7, re-sampling methods do not fix any of the issues induced by class imbalance for the logistic regression model.

Though we in Chapter 2 argued for splitting the data set into two disjoint

sets, namely a training and test set, we will now split the data set into three disjoint sets: training, validation and test set. The motivation behind this can be seen more clearly by looking closer at the current objective. Our goal is first to train m fitted models $\mathrm{M}_1, \ldots, \mathrm{M}_m$, then decide which of these m fitted models perform best, and third provide a figure for the predictive performance of our chosen model. The training set will be used to fit each of the m models, whose performance is then evaluated on the validation set. The method that performs best on the validation set is declared the best method. One could then report the predictive performance (for instance AUC) for the best fitted model computed on the validation set as the predictive performance of the best model. This seems unproblematic, since we have trained the model on one set, and evaluated the performance on an independent validation set as suggested in Chapter 2. However as pointed out by Hastie et al. (2009), using one data set to both select between models, and then secondly estimate model predictive performance on the same validation set is problematic. This is because we have actively selected that model for the sole reason that its performance was indeed superior on the validation set, and so the performance on the validation set tends to be biased due to this selection step. Therefore, when the final model has been selected, we estimate its performance with the third set, the test set. The training set will consist of all data from term 3 to term 33, validation set is from term 33 to term 35, and the test set is data in the terms 36 to 38. This division has been visualized in Figure 8.2. We have thus set aside 5 years worth of data for training, half a year for validation and another half for testing.

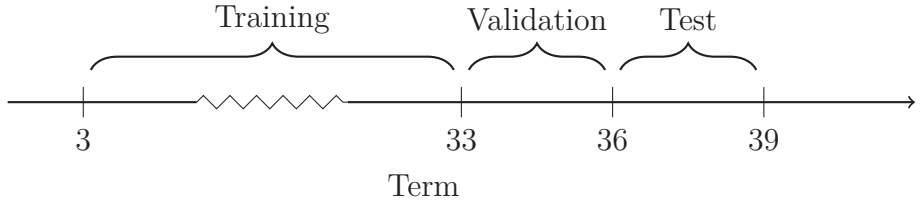


Figure 8.2: Illustration of the splitting into training, validation and test sets.

Care must be taken when performing data imputation not to extract any information from the test and validation sets into the training set. For this reason we did not include data from the test or validation sets to compute the imputed median values. Additionally, we imputed the test and validation sets using data solely from the training set.

Ridge

The ridge estimator will be fit using 10-fold cross validation repeated 5 times, with λ spaced equally between $(-9, 2)$ on the log scale. The results from the cross validation procedure for ridge is given in Figure 8.3. Each of the dashed lines represent one of the 5 cross validation estimates of AUC. The solid line is the average of these 5 cross validation estimates, which we will denote AUC_{cv}. We can see that for ridge regression there seems to be little variation in the AUC across folds, seeing that all the cross validation estimates of AUC are maximized at approximately the same λ.

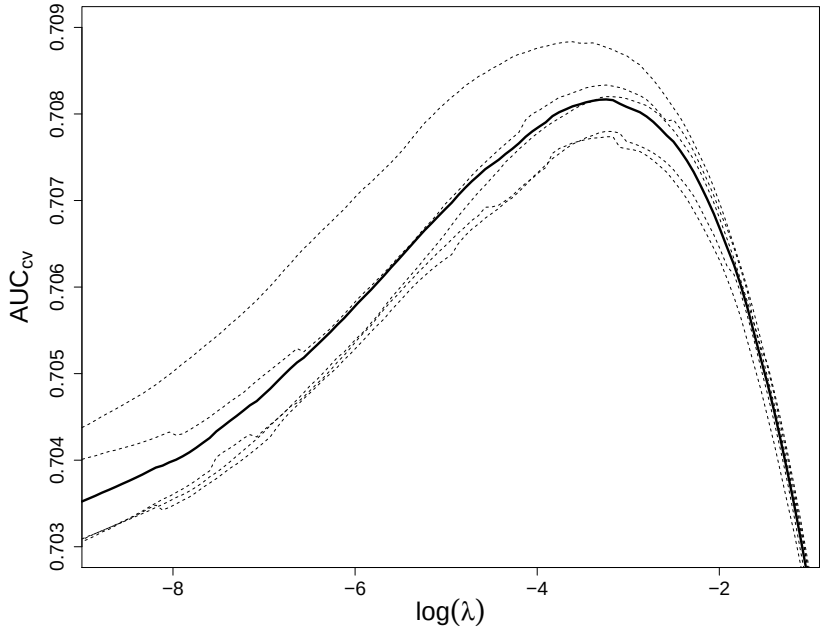


Figure 8.3: AUC_{cv} as well as the 5 cross validation estimates of AUC for the ridge penalty.

Lasso

The lasso estimator will be fit using 10-fold cross validation repeated 5 times, with lambda spaces equally between $(-10, -4)$ on the log scale. Figure 8.4 shows the same as Figure 8.3, but for the lasso penalty. There appears to be more uncertainty for lasso than ridge. It seems that one may in this case benefit more from repeated cross validation since the variance in AUC_{cv} is greater.

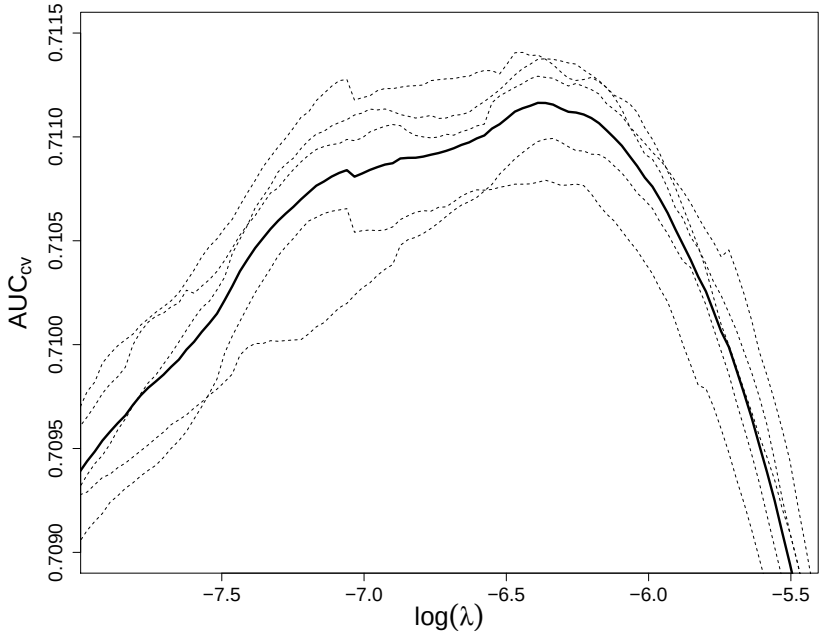


Figure 8.4: AUC_{cv} as well as the 5 cross validation estimates of AUC for the lasso penalty.

Elastic net

As with the two previous methods, the elastic net estimator will be fit using 10-fold cross validation repeated 5 times, with lambda spaced equally between $(-9, 2)$ on the log scale. The α parameter will be optimized over a grid between $(0.1, 0.9)$ with increments of 0.1. In addition, we add the possible α values $0.01, 0.05, 0.95, 0.99$ seeing that these were selected frequently in our simulation studies. Figure 8.5 is a surface plot of AUC_{cv} for values of both λ and α. Like in Figure 8.4, we also here see that AUC_{cv} is somewhat uneven, which is due to the variance of AUC_{cv}. We could have repeated the cross validation procedures more than 5 times in order to smooth out AUC_{cv} for lasso and elastic net. However this would induce increased computational cost which is already an issue when repeating the cross validation procedures 5 times. Of the three methods, elastic net is by far the most computationally demanding. Interestingly, choice of α does not appear to be very important as long as α is greater than approximately 0.3. AUC_{cv} seems to flatten out after this.

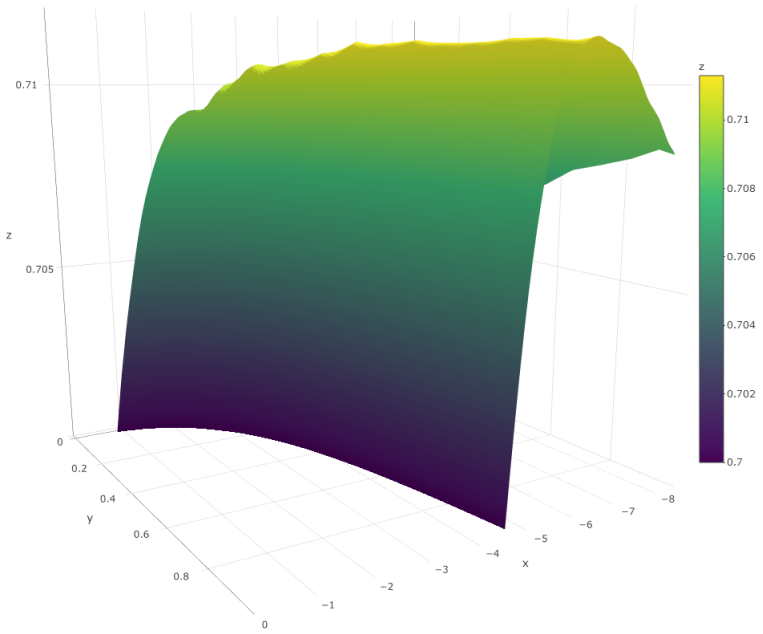


Figure 8.5: $\log(\lambda)$ values along the x-axis, α values along the y-axis, AUC_{cv} along the z-axis.

8.3 Results

The results for the ridge, lasso and elastic net applied to the VAT data set are given in Table 8.2. Beginning with the results for ridge, we see that AUC_{cv} is 0.7082. However, the AUC computed on the validation set AUC_{val} is higher. For lasso, AUC_{cv} is 0.7112, and the AUC for the validation set is 0.7125. Elastic gets an AUC_{cv} of 0.7112, the same as lasso, and an AUC_{val} of 0.7119. Lasso selects 255 covariates in the final model, lower than the elastic net which selects 411. For elastic net, the optimal α value is 0.3.

Penalty	$N_{\hat{\beta}\neq 0}$	AUC_{cv}	AUC_{val}	BS_{val}
Ridge	944	0.7082	0.7151	0.1631
Lasso	255	0.7112	0.7125	0.1627
Elnet (0.3)	411	0.7112	0.7119	0.1628

Table 8.2: Results for the three different methods, based on all data.

What is immediately apparent is the similarity of the predictive per-

formance for these three methods. Both in terms of AUC_{val} and BS_{val} the methods perform similarly. Ridge performs best on the validation set in terms of AUC, and should thus be selected as the method best suited if AUC is the desired model selection criterion. Lasso does however score a little better in Brier score. If we continue as we have previously in this thesis by using the AUC as the primary predictive performance measure, we would conclude that ridge is superior. To obtain an unbiased estimate of predictive performance we must then compute AUC and Brier score on the separate test set. This yields an AUC of 0.7055 and a Brier score of 0.1474, which are both lower than the same values computed on the validation set. For AUC this means that the model seems to be performing worse on the test set compared to the validation set, while Brier score shows the opposite.

Other measures of performance can be constructed for this application which may be of interest. Consider the problem at hand, namely that the Norwegian Tax Administration each year has a number of cases which should be controlled. This number is typically much larger than what is feasible to control, so we are trying to make the selection of cases to control more effective by assigning a probability of fraud to each case. The cases with the highest probability of fraud are then chosen for inspection. We can recreate this situation, by rank ordering all cases by their estimated probabilities and compute the proportion of fraudulent cases uncovered by inspecting say the top 10%. This has been done for a range of percentages, and the result is given in Figure 8.6. The figure shows the result of using the lasso fit to predict cases in the test and validation set. If our model were no better than chance, the proportion of fraudulent cases uncovered should equal the proportion of cases controlled. We see that by inspecting the top 5% according to their estimated probabilities, we have uncovered 13.8% of the fraudulent cases. Similarly, by inspecting the top 10%, we uncover 24.5% of the fraudulent cases. This percentage becomes less impressive as the proportion of cases controlled increases, with a mere 70% of fraudulent cases being among the top 50% ordered by probability of fraud.

Another related measure is the hit rate, that is, the number of fraudulent cases divided by the total number of cases. These can be computed for different ranges of the predicted probabilities, which was done in Berset et al. (2016). We wish to recreate the figure given there, but for our model and test data. Their model gave each case a score between 0 and 100 indicating the probability of fraud. They then plotted the hit rate for cases with scores in the ranges (90,100), (80,89) and (75,79). We can use the same approach by multiplying our predicted probabilities by 100. Unfortunately,

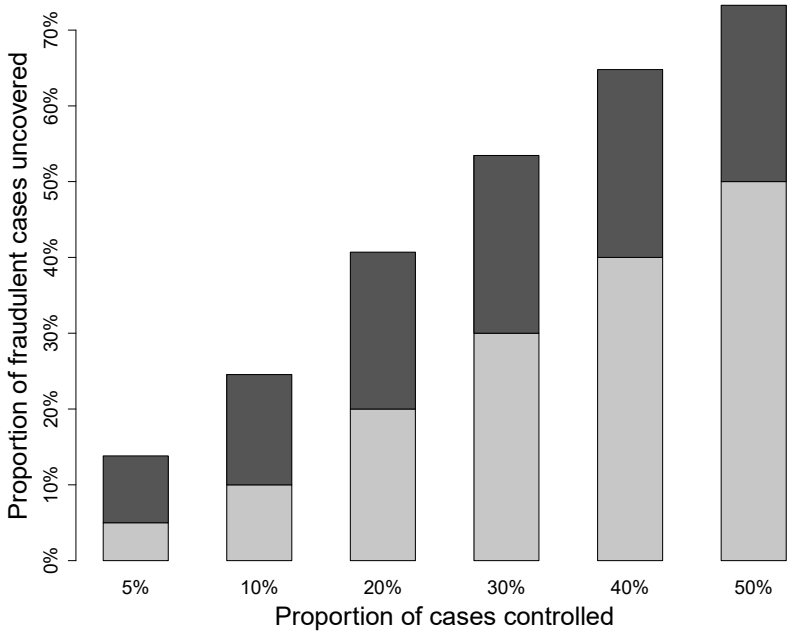


Figure 8.6: Proportion of the fraudulent cases uncovered by inspecting top percentages of the estimated probabilities. Grey portion corresponds to random guessing, black portion illustrates the added performance gain of our fitted model.

the resulting figure is not very informative in our case because only a few cases are assigned probabilities in the range (0.9,1). For this reason, we choose to look at the ranges of the percentiles of the predicted probabilities, see Figure 8.7. We have also for this figure used the lasso fit and predicted on the data from the test and validation sets. Of those controls where the predicted probabilities were in the 75'th to 79'th percentiles, roughly 30% of the cases were fraudulent. When this range is increased to 80'th to 89'th percentile we see a small increase to about 36%. A greater increase is observed for the upper 10'th percentile, where more than half of the controls were actually fraudulent. The performance of the methods used for selecting cases currently employed at the Norwegian Tax Administration is not known. Hence, it is difficult to make conclusions about whether the performance of our model offers any improvement.

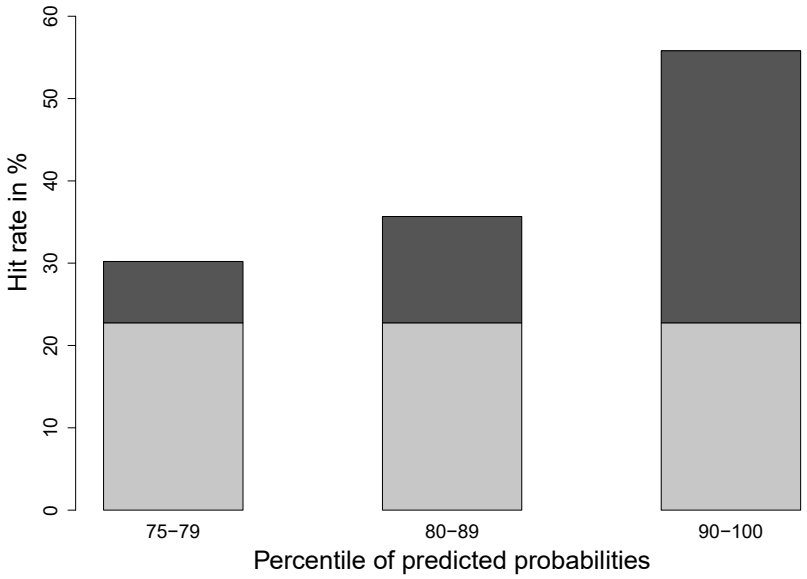


Figure 8.7: Hit rate for ranges of percentiles of the predicted probabilities. Grey portion corresponds to random guessing, black portion illustrates the added performance gain of our fitted model.

8.4 Modeling fraud over time

In the previous section we took a somewhat static view of the fraud detection problem by training models using one data set, and testing on data from the same time period for all models. We would also like to test which method performs best over time, thus reducing our reliance on a small part of the data set (the validation set) for selection of the final model. This is interesting because it recreates the way in which fraud models will be used, with frequent updating on more and more data as time progresses.

We will now train logistic regression models at time points t_k, for $k = 1, \ldots, K$. For each time $t_k, 0 < k < K$ we predict the cases between times t_k and t_{k+1}. Optimally, we would set $K = 38$ such that we re-train the models for the new data given in each term. Prediction would then be performed only on the single next term. This is however numerically demanding since it requires training of a large number of models. Instead, we propose to set $K = 6$, and define the times t_k such that $t_0 = 3$, $t_1 = 8$, $t_2 = 14$, $t_3 = 20$, $t_4 = 26$, $t_5 = 32$, $t_6 = 38$. That is, we increment time by 6

terms, which is one year. So, for the first sets of models trained at t_1 will use data from the terms 3 to 8 as training data, and then test on the terms 9 through 14. The next training time point t_2 trains on data from terms 3 through 14, and tests on the terms 15 through 20, and so on. The AUC_{test} will be given for each term, and not averaged over the time periods. See Table 8.3 for a full overview of which terms are included in the training and test sets at each time point, as well as some information about the training set at each time point.

Time	Train terms	Test terms	Nr. of cases	Fraudulent cases
t_1	3:8	9:14	6 820	1 427
t_2	3:14	15:20	16 980	3 427
t_3	3:20	21:26	26 522	5 525
t_4	3:26	27:32	36 049	7 199
t_5	3:32	33:38	43 988	8 978

Table 8.3: Overview of data sets at each time point.

One must also remember that for this analysis to be appropriate we can not use the imputed data set from the previous section. In that data set we defined data from term 3 to term 33 as training data, and thus all imputed values are computed based on data between these time points. However, we now wish to test on data that are in between these two time points. This is problematic because we do not want to use data from the test set to impute values in the training set. So we must create one training and one test set for each of the time points $t_k, 0 < k < K$, and impute missing values accordingly.

The results of applying the three methods as described above is shown in Figure 8.8. AUC computed on the test set is shown for lasso in red solid line with red circles and ridge in blue dashed line with angled squares, and elastic net in dotted lines with black squares. A number of conclusions can be drawn from this plot. First, we see that lasso performs better than ridge on average, and slightly better than elastic net. While the differences in AUC between lasso and ridge can be (relatively) large, the results for elastic net are usually somewhere in between these two. After term 20, elastic net seems to perform similarly to lasso. This behavior seemed curios, and was checked by running elastic net several times, but the results were always as shown here. Figure 8.8 provides additional insight into the performance

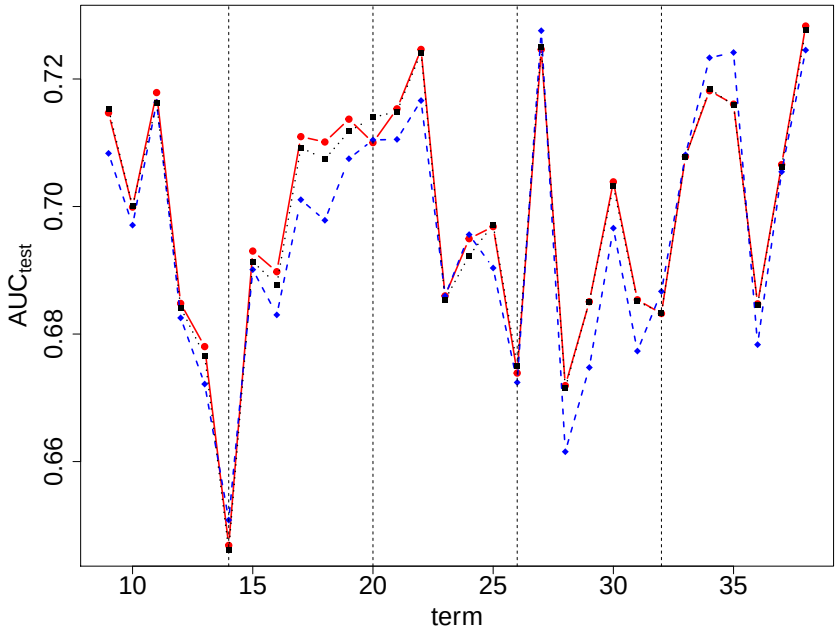


Figure 8.8: AUC_{test} computed over time for lasso (red circles) and ridge (blue angled squares), and elastic net (black squares) fits. Vertical dashed lines indicate model updating times.

of the three methods compared to that provided by Table 8.2. We can see that the data chosen as our validation set for creation of Table 8.2, which corresponds to the data from the first three terms after the last dashed vertical line in Figure 8.8, was perhaps unfortunate. This is because at least the 2nd and 3rd terms after the last vertical line are in fact one of the very few terms in which ridge performs better than elastic net and especially lasso. The results from Figure 8.8 at least begs the question of whether ridge actually provides the best predictive performance. Having utilized the data to a higher degree by performing such a sequential analysis we see that the ridge fit may in fact be sub-optimal, and that maybe either lasso or elastic net should be preferred instead. The variable selection properties of these two latter methods, and particularly elastic net as demonstrated in Chapter 6, is another sound argument to consider either of these methods over ridge.

Another purpose of this analysis was to study whether there was any indication of either model improvements or deterioration with respect to AUC as time progressed. One might suspect that predictive performance would improve as more data became available, but this does not seem to be

the case. On the other hand, it is possible that any model trained on data from the first terms would be biased in its predictions on data from the last terms, due to a changing nature of the data over time. Such changes could be induced by changes in operations or data collection of The Norwegian Tax administration. By again returning to Figure 8.1 we can see that there may be signs of seasonality, since the class balance reaches a peak every 6 terms. This seasonality does not appear to be present in the last year of data, so this may be reason to suspect that there may be some time dynamic making prediction on future data points more difficult.

8.5 Chosen covariates

Other than looking at predicitve performance, it is also interesting to see how the estimated $\boldsymbol{\beta}$ coefficients are distributed. The following results are obtained by using all available training data as was the case when creating Table 8.2. Ridge seemed to offer the best AUC on the validation set, while lasso performed best on the data set over time, as illustrated in Figure 8.8. Additionally, lasso unlike ridge does perform variable selection. For this reason we will consder the fit of the logistic regression model obtained by lasso. One could argue that the results of our simulations conducted in Chapter 6 also should be taken into account when deciding on the final method. If so, then one could argue that elastic net demonstrated arguably superior variable selection performance over lasso, and should therefore be chosen here. However, the number of variables included by elastic net in this case is quite large (411) and in addition elastic net also performs somewhat worse than lasso in terms of AUC. Another point to consider is that elastic net struggles selecting the value for α, as demonstrated by the flatness of AUC_{cv} in Figure 8.5 for α values greater than 0.3. The apparent indifference of elastic net in the choice of α (as long as $\alpha \geq 0.3$) introduces additional variability into the distribution of $\hat{\boldsymbol{\beta}}$.

We begin by looking at the coefficients for the numerical covariates. In order to make the comparison independent of the scale of the covariates, we have standardized all coefficients. This is done by multiplying the unstandardized $\hat{\boldsymbol{\beta}}$ values by the standard deviation of the relevant covariate. The 10 largest standardized coefficients in absolute value are shown in Figure 8.9. The color of each bar represents the sign of the coefficient: black means negative, grey means positive. Recall that in our model a negative coefficient reduces the probability of fraud, and a positive coeffient increases the probability of fraud. Because the data are anomymized, there is not much

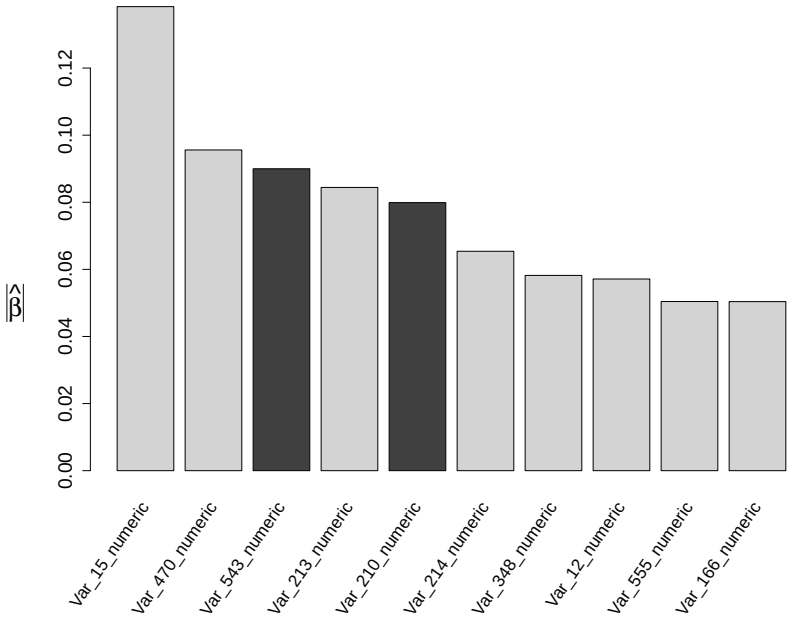


Figure 8.9: The 10 largest coefficients for numerical covariates.

that can be said. However, there does appear to be some covariates that are more imporant than others. Figure 8.10 shows the same as Figure 8.9 but for the categorical covariates. Note that these have, as previously explained, been converted to binary covariates which is what Figure 8.10 displays. We see that the indicator variables we constructed to indicate when a numerical value was missing seem to be quite important. Some of these coefficients, for instance "Var_6_numericMISSING" have a negative sign, meaning that if the covariate "Var_6_numeric" is missing, then the estimated probability of fraud decreases. Again, interpreting these coefficients would have been easier had the true covariate names been known. Additionally, *Var_81* seems to be important, with the coefficients of two of its levels appearing among the top 10.

We can also compute the number of covariates selected in the four covariate categories: *background*, *current assignment*, *previous terms* and *previous years*, see Table 8.4. In the largest group *previous terms* 80 covariates were selected thus being the group from which most variables were selected. However, it is also the group with the lowest percentage of covariates chosen. The group *current assignment* followed by *background* scores best in this respect, with 50% and 41% of the covariates chosen, respectively. This is

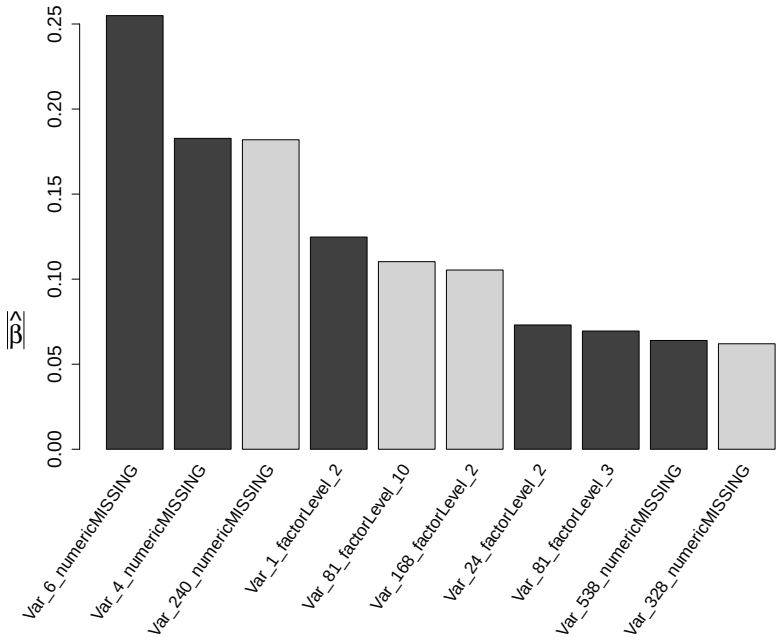


Figure 8.10: The 10 largest coefficients for binary covariates.

	Background	Current assignment	Previous terms	Previous years
Total	168	42	220	126
Selected	69	21	80	48
Percentage	41%	50%	36%	38%

Table 8.4: Distribution of the selected covariates among the four different groups.

perhaps not surprising, seeing that information about the business being controlled, as well as the facts regarding the current control should be able to explain much of the variation. Still, we see that covariates from the two other groups certainly contribute to the final model.

8.5.1 Measuring stability of the lasso fit

As mentioned in Chapter 3 lasso can be variable in terms of variable selection when covariates are highly dependent. As we have seen in Chapter 5 many covariates in the VAT data set are indeed affected by dependence. Until now we have studied the importance of covariates based on their absolute

value for one fit of lasso. Due to the high levels of dependence between covariates, this is perhaps an overly simplistic picture since it does not provide a picture of the uncertainty involved. If some of the covariates given in Figures 8.10 and 8.9 are highly dependent with other covariates, then one would expect that they could easily be replaced by any of those highly dependent covariates in another lasso fit. For this reason we will now study the uncertainty in the variable selection procedure. This means we will measure the uncertainty related to the tuning of the λ parameters as a result of the 10-fold cross validation procedure. The large uncertainty coming from the randomness of the data is thus not taken into account in the following.

The 10 fold stratified cross-validation procedure has been repeated 200 times, each time with different partitioning of the folds. We thus obtain 200 different λ values, which will result in 200 lasso fits. Figure 8.11 shows the number of covariates selected in these fits in a histogram. In addition to presenting the frequencies, we also provide the estimated density using the `density()` function in R. There are some gaps in the histogram, which is due to a low number of unique observations. In fact, out of 200 repetitions we are left with only 19 unique values. Certainly, some degree of uncertainty is present, but it seems that our lasso fit which selected 255 covariates is quite close to the average number (272) of covariates selected. Figure 8.11 also illustrates that repeating the cross validation procedure was a good idea because of the large variance in the number of covariates. All the 10 covariates shown in each of the Figures 8.9 and 8.10 are selected in all 200 lasso fits. This suggests that those covariates the model emphasizes most are perhaps not affected by dependence with other covariates. This result also strengthens the choice of the lasso model, seeing that in this case we actually obtain a seemingly stable fit. This is strengthened further by Figure 8.12 which shows the frequency of the covariates that occurred most often in the 200 lasso fits. 187 were selected in all 200 fits, and 465 covariates were selected in one or more of the 200 fits.

8.6 Summary

In this chapter we analyzed the VAT data set which we have referred to in both Chapter 1 and 5. Data belonging to the beginning and end points (in time) of the data set was removed. We then imputed the missing values by median imputation and extracted further information from the data by recording the instances of missing data. The data set was then increased to a total of 944 covariates. We then applied some of the methods for

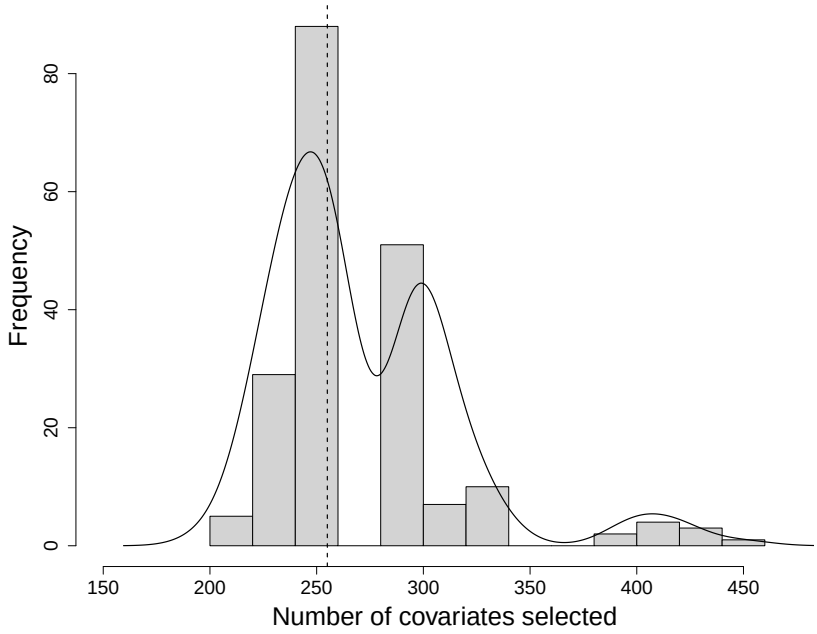


Figure 8.11: Results from repeating the 10 fold cross validation procedure 200 times. Dashed vertical line represents number of covariates selected for the lasso model in Section 8.5.

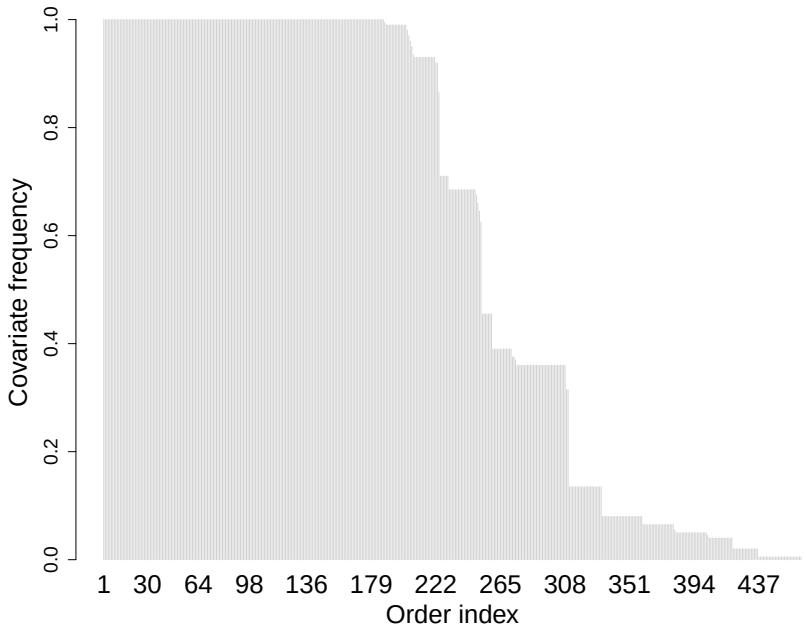


Figure 8.12: Ordered (descending) frequencies of selected covariates.

regularization discussed in Chapter 3 to these data. Only ridge, lasso and elastic net were tried because these illustrated promising behavior in the simulations in Chapter 6. All three methods performed similarly in terms of AUC, but yielded different fitted models regarding how many covariates were included. Ridge of course uses the highest number of covariates since it does not perform variable selection. Elastic net included 411 covariates which is approximately 43% of all covariates in the data set. We also saw that elastic net struggled finding a uniquely best α value, as shown in Figure 8.5. As long as $\alpha \geq 0.3$ the choice of α seemed to be not very important. This introduces much uncertainty regarding the elastic net model since α has a large impact on the final coefficients. Additionally the computational effort is much larger for elastic net than ridge or lasso, which is definitely a disadvantage. Seeing that the predictive performance of lasso in AUC was similar to, if not better than elastic net we therefore opted for the lasso model which selected 255 covariates. All the four groups of covariates were represented among these 255, with the groups *background* and *current assignment* scoring the highest percentage of covariates included. In addition to modeling the data in a somewhat static sense by training on the first 5 years and testing on the last year, we also trained our models over time, always predicting on future data. The results showed that lasso indeed did achieve the best AUC over time. There were of course fluctuations in AUC, but there was no clear improvement to AUC as more data became available for training.

The fact that lasso achieves at least as good performance compared to ridge and elastic net, but uses much fewer covariates can be an advantage to model interpretation. Perhaps more important, it means that the model fit using lasso requires much less data to be gathered about each control than does a model fit using ridge or elastic net. This is beneficial because it reduces the amount of data that needs to be gathered. Reducing the number of coefficients also reduces the amount of work and resources necessary for storing and upkeep of such data sets.

Chapter 9

Conclusion and discussion

The end goal of this thesis was to model VAT fraud probabilities in the data set provided by the Norwegian Tax administration. We illustrated that one problem with this data set was the large degree of dependence between covariates. We therefore conducted a simulation experiment where we focused particularly on recreating the dependence structures between the covariates in the VAT data set. Generating dependent covariates from different families of distributions, both numerical and categorical was of some concern initially. However, we solved the problem by generating data using a copula approach. This approach proved to be very flexible, because we can construct complicated dependence structures with no restrictions on the marginal distributions for the covariates. Using this setup, we then conducted a simulation experiment where the goal was to study which methods for regularization for logistic regression perform best when covariates are highly dependent. We found that elastic net performed well in terms of prediction and arguably best in terms of variable selection. Elastic net included a large proportion of the true non-zero covariates in the model, depending on the $\boldsymbol{\beta}$ coefficients. Lasso also performed well for prediction, though sometimes not as good as elastic net, again depending on the $\boldsymbol{\beta}$ coefficients in the true model. Lasso was generally more conservative in its variable selection than elastic net. We also provided an explanation to why the AUC increased as the correlation among the non-zero covariates increased in Chapter 6. We hope that this insight can be of use to others when designing similar simulation studies. One possible extension of our work could be to study how using different copulas to model dependence between covariates would affect variable selection and prediction. If data such as the VAT data set are available, one can select an appropriate copula by maximum pseudolikelihood estimation, as described in Genest and Favre (2007). They focus mainly on bivariate copulas, but extending their work to

the p-dimensional case, we get the following procedure: Define

$$F_n^j(\mathrm{x}) = \frac{1}{n+1}\sum_{i=1}^{n} I(\mathrm{x} \geq \mathrm{x}_{i,j}),$$

for $j = 1, \ldots, p$, and maximize the pseudolikelihood given by

$$\prod_{i=1}^{n} c\left(F_n^1(\mathrm{x}_{i,1}), F_n^2(\mathrm{x}_{i,2}), \ldots, F_n^p(\mathrm{x}_{i,p})\right),$$

where c is the density of some p-dimensional copula. Not only could one then use the pseudolikelihood to select the copula that fits the data best, but also use it to estimate parameters, in our case the correlation matrix $\boldsymbol{\rho}$. New simulation studies could then be conducted using the resulting estimated copula for modeling dependence between covariates.

The effect caused by class imbalance on the predictive performance of a logistic regression model was unclear prior to our analyses. Similarly, the effects this has on variable selection has to our knowledge not been studied previously. A second objective of this thesis was therefore to provide some clarity on these issues. We observed that the AUC of our logistic regression models did not suffer as much as did random forest when the class balance was reduced. However, the performance of random forest could be improved markedly by using any of the re-sampling methods, most notably under-sampling. Logistic regression did not seem to benefit from such re-sampling methods. The variable selection capabilities of both lasso and elastic net worsened when the class balance was reduced, and particularly elastic net was highly variable in the number of covariates included in the model. The re-sampling methods did not seem to offer any improvements to variable selection either. We therefore recommend using lasso with no re-sampling when dealing with unbalanced data for logistic regression.

Based on the conclusions drawn from the simulation studies we progressed to analyze the VAT data set. Logistic regression models were fit using ridge, elastic net and lasso for regularization. Ridge performed well in AUC in the last part of the data set, but did not score as high as elastic net and lasso on data from earlier time points. Predictive performance was similar between lasso and elastic net. The AUC of the model fit by lasso was approximately 0.70, depending on which part of the data set one selects as the test set. Our work demonstrates that penalized logistic regression can be used to improve VAT fraud detection. However, we have not studied whether there are any signs of non-linearity in the VAT data set. A natural extension of

the analyses conducted in this thesis would be to consider also generalized additive models. For logistic regression, the linear predictor then becomes (Hastie et al., 2009)

$$\eta(\mathbf{x}_i) = \beta_0 + \sum_{j=1}^{p} f_j(\mathrm{x}_{i,j}),$$

which introduces a high degree of flexibility on the linear predictor $\eta(\mathbf{x}_i)$ due to the unspecified nature of the functions f_j for $j = 1, \ldots, p$. This in turn allows for a more flexible model for $P(Y_i = 1|\mathbf{x}_i)$. One problem with such additive models is that they may perform poorly when the number of covariates is high, depending on how the model is fitted. However, methods for fitting additive models have been proposed to alleviate this problem, see Tutz and Binder (2006).

Missing data was present to a large degree in the VAT data set. In addition, many of the covariates were highly dependent. It could therefore be worthwhile to study whether a more comprehensive imputation method than median imputation could improve predictive performance. One possible approach would be to perform a regression analysis on one covariate at a time, using the remaining covariates as explanatory variables (Hastie et al., 2009). One could then predict the missing values. Such an approach would take into consideration any structure among the covariates when imputing, which median imputation does not.

It should also be noted that the final analysis of the VAT data set was restricted by the anonymity of the data set. A more in-depth analysis would have been easier to carry out had the covariate names been known. For instance, it is easier to create interaction terms that are more likely than others to be informative when covariate names are known. Similarly, knowing the covariate names can help us in implementing non-linear effects of some covariates. Interpretation and evaluation of the final model would also have been easier if the covariate names had been known.

Appendix A

R-code

A.1 Chapter 2

Create stratified folds for k-fold cross validation.

```
stratified <- function(X,Y,k){
  #     X - Covariates
  # Y - Response
  # k - nr. of folds

  #find indexes which contains 1's and 0's
  ind_1 <- which(Y==1)
  ind_0 <- which(Y==0)

  #find number of 1's and 0's
  n_1 <- length(ind_1)
  n_0 <- length(ind_0)

  n <- n_1 + n_0

  floor_1 <- floor(n_1/k)
  floor_0 <- floor(n_0/k)

  #how many left?
  left_1 <- n_1-k*floor_1
  left_0 <- n_0-k*floor_0

  labels <- rep(0,n)

  #sample fold labels for 1's
  tags_1 <- rep(1,floor_1)%*%t(1:k)
  if(left_1 == 0){
    labels[ind_1] <- sample(c(tags_1))
  }else if(left_1 > 0){
```

```
    labels[ind_1] <- sample(c(tags_1,1:left_1))
  }

  #sample fold labels for 0's
  tags_0 <- rep(1,floor_0)%*%t(1:k)
  if(left_0 == 0){
    labels[ind_0] <- sample(c(tags_0))
  }else if(left_0 > 0){
    labels[ind_0] <- sample(c(tags_0,1:left_0))
  }
  return(labels)
}
```

A.2 Chapter 5

Produce the correlation matrix $\boldsymbol{\rho}$ as outlined in Chapter 6.

```
produceRho <- function(p_g,n_g,high,med,low,constant_cor,high.min){
  # p_g - number of covariates in group
  # n_g - number of groups
  # high - high level dependence value
  # med - medium level dependence value
  # low - low level dependence value
  # constant_cor - should decaying dependence be applied to the high-
      level dependence block?
  # high.min - the lowest level of within-group dependence

  p <- p_g*n_g

  if(p_g==10 && n_g==100){
    #groups number 90-100 are independence groups
    independence_group_start <- 90
  }else if(p_g==20 && n_g==50){
    #groups number 45-50 are independence groups
    independence_group_start <- 45
  }
  #initialize rho
  rho <- diag(p)
  #Define high,medium and low level matrices
  block_high <- matrix(high,p_g,p_g)
  block_med <- matrix(med,p_g,p_g)
  block_low <- matrix(low,p_g,p_g)

  lower_high <- high.min

  #compute decay weights
  w <- ifelse(constant_cor,0,-log(lower_high/high)/(20-2)*0.99)
```

```
30   #Compute the high level matrix with decay to correlations
31   for(i in 1:p_g){
32     block_high[i,] <- block_high[i,]*exp(-w*(abs(1:p_g - i) - 1))
33   }
34
35   diag(block_high) <- 1
36   #Insert the high, medium and low level block matrices into the
         correct indexes of rho
37   for(i in (1:(n_g))){
38     ind_x <- 1 + (i-1)*p_g
39     ind_y <- 1 + (i-1)*p_g
40
41     if(i <= independence_group_start){
42       rho[ind_x:(ind_x+p_g-1),ind_y:(ind_y+p_g-1)] <- block_high
43     }else if(i > independence_group_start){
44       rho[ind_x:(ind_x+p_g-1),ind_y:(ind_y+p_g-1)] <- diag(p_g)
45     }
46
47     if(i > 1 && i <= independence_group_start){
48       rho[(ind_x - p_g):(ind_x - 1),ind_y:(ind_y+p_g-1)] <- block_med
49     }
50     if(i > 2 && i <= independence_group_start){
51       rho[(ind_x - 2*p_g):(ind_x - p_g - 1),ind_y:(ind_y+p_g-1)] <-
             block_low
52     }
53   }
54
55   #The above for loop only defined the upper triangle of rho. The
         following creates the full matrix
56   rho[lower.tri(rho)] <- t(rho)[lower.tri(rho)]
57   rho
58 }
```

Generate data using the copula approach in Chapter 5.

```
1  gen_data <- function(n,p_g,n_g,beta0,beta,high,med,low,constant_cor,
       high.min){
2
3    p <- p_g*n_g
4    #produce the correlation matrix
5    rho <- produceRho(p_g=p_g,n_g=n_g,high=high,med=med,low=low,constant
         _cor=constant_cor,high.min=high.min)
6
7    #If matrix is not pos. definite break.
8    if(sum(eigen(rho)$values<0)>0){
9      cat("Specified correlation matrix is not positive definite!")
10   }else{
11     #Initialize X
12     X <- matrix(0,ncol=p,nrow=n)
13     #Generate n instances of p correlated normal variables.
```

```
14     Z <- mvrnorm(n=n,rep(0,p),rho)
15     #Transform from Z to U using cumulative density function.
16     U <- pnorm(Z)
17
18     #Transform the marginals. Done as described in Chapter 6. The
           following code is difficult to read because it needed to be
           vectorized to reduce computational cost as commented in Chapter
            6. It is essentially just a matter of extracting the correct
           elements of U and transforming these using the correct
           marginals, and inserting them into the correct position into X.
19     for(i in 1:n){
20       if(p_g == 10 && n_g == 100){
21         X[i,1:3 + c(rep(p_g,3)%*%t(0:(n_g-1)))] <- qnorm(U[i,1:3 + c(
               rep(p_g,3)%*%t(0:(n_g-1)))])
22         X[i,4:6 + c(rep(p_g,3)%*%t(0:(n_g-1)))] <- qgamma(U[i,4:6 + c(
               rep(p_g,3)%*%t(0:(n_g-1)))],1,1)
23         X[i,7:10 + c(rep(p_g,4)%*%t(0:(n_g-1)))] <- ifelse(U[i,7:10 +
               c(rep(p_g,4)%*%t(0:(n_g-1)))]<0.5,0,1)
24       }else if(p_g == 20 && n_g == 50){
25         X[i,1:9 + c(rep(p_g,9)%*%t(0:(n_g-1)))] <- qnorm(U[i,1:9 + c(
               rep(p_g,9)%*%t(0:(n_g-1)))])
26         X[i,10:12 + c(rep(p_g,3)%*%t(0:(n_g-1)))] <- qgamma(U[i,10:12
               + c(rep(p_g,3)%*%t(0:(n_g-1)))],1,1)
27         X[i,13:20 + c(rep(p_g,8)%*%t(0:(n_g-1)))] <- ifelse(U[i,13:20
               + c(rep(p_g,8)%*%t(0:(n_g-1)))]<0.5,0,1)
28       }
29     }
30   }
31   #Compute linear predictor
32   eta <- beta0 + X%*%beta
33   #Compute probability of Y=1
34   P <- exp(eta)/(1+exp(eta))
35   #Sample from Bernoulli distribution with probability P.
36   Y <- rbinom(n,1,p=P)
37   return(list(Y=Y,X=X,eta=eta))
38 }
```

A.3 Chapter 6

Create cross validation procedure for SCAD using AUC as performance measure.

```
1 cv_scad <- function(X,Y,k,alpha,lambda_sequence){
2   #create stratified folds
3   cv_ind <- stratified(X,Y,k=k)
4
5   #initialize
```

```
  cv_AUC <- matrix(0,nrow=k,ncol=length(lambda_sequence))
  #for each iteration in the cv routine
  for(i in 1:k){
    #train model for current training set, and return AUC on the
        current test set
    cv_AUC[i,] <- scad_auc(X_train = X[-which(cv_ind==i),],
                           Y_train = Y[-which(cv_ind==i)],
                           X_test = X[which(cv_ind==i),],
                           Y_test = Y[which(cv_ind==i)],
                           alpha = alpha,
                           lambda_sequence = lambda_sequence)

  }
  return(list(cv_AUC=cv_AUC,lambda_sequence=lambda_sequence))
}

scad_auc <- function(X_train,Y_train,X_test,Y_test,alpha,lambda_
    sequence){
  number_of_runs <- length(lambda_sequence)
  AUC_v <- 1:number_of_runs*0

  #NOTE#
  #The following approach, i.e. fitting one lambda value sequentially,
        is slow but necessary due to the instability of "ncvreg" for
      small values of lambda
  for(i in 1:number_of_runs){
    scad_model <- ncvreg(X_train,Y_train,family="binomial",penalty="
        SCAD",gamma=alpha,lambda=lambda_sequence[i])
    Y_pred <- predict(scad_model,X_test,type="response")
    #compute AUC
    AUC_v[i] <- AUC(Y_test,Y_pred)$AUC
  }

  return(AUC_v)
}
```

Script for running simulations in Chapter 6. This is the full version of the pseudocode given in Algorithm 6.1.

```
#my own package, contains functions like "stratified" given above
library(package)
library(ncvreg)
library(MASS)
library(glmnet)
library(DMwR)
library(doParallel)

study1 <- function(m,regularization,cores,simulation){
 #initialize parameter values
```

```
11  coeff_type <- "constant"
12  high <- 0.9
13  spacing <- 0
14  high.min <- 0.7
15  p_g <- 10
16  n_g <- 100
17  n <- 3000
18  if(simulation==1){
19    med <- 0
20    low <- 0
21    constant_cor <- T
22  }else if(simulation==2){
23    med <- 0
24    low <- 0
25    constant_cor <- F
26  }else if(simulation==3){
27    med <- 0.4
28    low <- 0.2
29    constant_cor <- F
30  }else if(simulation==4){
31    med <- 0.4
32    low <- 0.2
33    constant_cor <- F
34    spacing <- 1
35  }else if(simulation==5){
36    coeff_type <- "varied"
37    med <- 0.4
38    low <- 0.2
39    constant_cor <- F
40    spacing <- 1
41  }else if(simulation==6){
42    coeff_type <- "varied"
43    med <- 0.4
44    low <- 0.2
45    constant_cor <- F
46    spacing <- 1
47    p_g <- 20
48    n_g <- 50
49  }else if(simulation==7){
50    coeff_type <- "varied"
51    med <- 0.4
52    low <- 0.2
53    constant_cor <- F
54    spacing <- 1
55    p_g <- 20
56    n_g <- 50
57    n <- 30000
58  }
59
```

```
60  #Produce the beta-vector. The function "produceBeta" is in the
        package "package" referred to above.
61  tmp_res <- produceBeta(p_g=p_g,n_g=n_g,spacing=spacing,coeff_type=
        coeff_type)
62
63  #get beta_0 value
64  beta0 <- tmp_res$beta0
65  #get beta vector
66  beta <- tmp_res$beta
67
68  #initialize cluster
69  cl <- makeCluster(cores)
70  registerDoParallel(cl)
71  #Run for-loop in parallel
72  parallel_result <-
73  foreach(i=1:m,.combine='rbind',.packages=c('MASS','glmnet','ncvreg',
        'package','DMwR'),.verbose=TRUE) %dopar% {
74        hyper_param <- -1
75
76    ######################
77    ### Generate data ###
78    ######################
79    data <- gen_data(n=(n+3000),p_g=p_g,n_g=n_g,beta0=beta0,beta=beta,
          high=high,med=med,low=low,constant_cor=constant_cor,high.min=
          high.min)
80    data_train <- list(Y=data$Y[1:n],X=data$X[1:n,])
81    data_test <-  list(Y=data$Y[(1+n):(n+3000)],X=data$X[(1+n):(n
          +3000),])
82
83    #Cross validation fold id's
84    labels <- stratified(data_train$X,data_train$Y,k=10)
85
86    nlambda <- 50 #default is 100
87    ####################
88    ### Train model ###
89    ####################
90    if(regularization=="scad"){
91      #Need to supply lambda values for SCAD
92      if(n==30000){
93        lambda_sequence <- exp(seq(-6.5,-2,length=nlambda))
94      }else{
95        lambda_sequence <- exp(seq(-5.5,-2,length=nlambda))
96      }
97      #Note: "gamma" here equals the "a" parameter for SCAD in Chapter
            3.
98      #Option to optimize over gamma as well
99      gamma_sequence <- 3.7
100     tmp_result <- lambda_opt <- 1:length(gamma_sequence)*0
101
```

```
102       for(j in 1:length(gamma_sequence)){
103         tmp_mod <- cv_scad(X=data_train$X,Y=data_train$Y,k=10,alpha=
              gamma_sequence[j],lambda_sequence=lambda_sequence)
104         tmp_result[j] <- max(apply(tmp_mod$cv_AUC,2,mean))
105         lambda_opt[j] <- lambda_sequence[which.max(apply(tmp_mod$cv_
              AUC,2,mean))]
106       }
107           #Find optimal gamma
108       gamma_final <- gamma_sequence[which.max(tmp_result)]
109       #Find optimal lambda
110       lambda_final <- lambda_opt[which.max(tmp_result)]
111       #Fit final model
112       final <- ncvreg(data_train$X,data_train$Y,family="binomial",
            penalty="SCAD",lambda=lambda_final,gamma=gamma_final)
113       hyper_param <- gamma_final
114       betahat <- final$beta[-1]
115
116     }else if(regularization=="lasso"){
117       cvresult <- cv.glmnet(data_train$X,data_train$Y,family="binomial
              ",alpha=1,type.measure="auc",foldid=labels,nlambda=nlambda)
118       betahat <- as.numeric(coef(cvresult,s="lambda.min"))[-1]
119
120     }else if(regularization=="ridge"){
121       cvresult <- cv.glmnet(data_train$X,data_train$Y,family="binomial
              ",alpha=0,type.measure="auc",foldid=labels,nlambda=nlambda)
122       betahat <- as.numeric(coef(cvresult,s="lambda.min"))[-1]
123
124     }else if(regularization=="elnet"){
125       alpha_sequence <- c(0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.95,0.99)
126       tmp_result <- lambda_opt <- 1:length(alpha_sequence)*0
127       models <- list()
128       #optimize over alpha
129       for(j in 1:length(alpha_sequence)){
130         models[[j]] <- cv.glmnet(data_train$X,data_train$Y,family="
              binomial",alpha=alpha_sequence[j],type.measure="auc",foldid
              =labels,nlambda=nlambda)
131         tmp_result[j] <- max(models[[j]]$cvm)
132         lambda_opt[j] <- models[[j]]$lambda.min
133       }
134       cvresult <- models[[which.max(tmp_result)]]
135
136       #find best alpha
137       alpha_final <- alpha_sequence[which.max(tmp_result)]
138
139       hyper_param <- alpha_final
140       betahat <- as.numeric(coef(cvresult,s="lambda.min"))[-1]
141
142     }else if(regularization=="alasso"){
143       lambda_sequence <- rev(exp(seq(-10,1,length=nlambda)))
```

```
144
145       folds <- stratified(data_train$X,data_train$Y,k=10)
146
147           #Cross validation for adaptive lasso.
148           #This ensures that the weights are computed for each
                  iteration. See chapter 6.
149           #The method "cv_glmnet" is defined below.
150       tmp_mod <- cv_glmnet(X=data_train$X
151                            ,Y=data_train$Y
152                            ,folds=folds
153                            ,re_method="no"
154                            ,re_balance=0
155                            ,alpha=1
156                            ,weights_gamma=1
157                            ,lambda_sequence=lambda_sequence)
158
159       cv_auc <- apply(tmp_mod$cv_AUC,2,mean)
160       optimal_lambda <- lambda_sequence[which.max(cv_auc)]
161
162       ##############################
163       ### COMPUTE FINAL WEIGHTS ###
164       ##############################
165
166       weights_cvresult <- cv.glmnet(data_train$X,data_train$Y,family="
              binomial",alpha=0,type.measure="auc",foldid=folds,nlambda=
              nlambda)
167       beta_values <- as.numeric(coef(weights_cvresult,s="lambda.min"))
              [-1]
168
169       weights_final <- abs(1/beta_values)
170       weights_final[weights_final==Inf] <- 999999
171           #train final model using correct weights
172       cvresult <- glmnet(data_train$X,data_train$Y,family="binomial",
              alpha=1,penalty.factor=weights_final,lambda=lambda_sequence)
173
174       betahat <- as.numeric(coef(cvresult,s=optimal_lambda))[-1]
175       hyper_param <- 1
176     }else if(regularization=="no"){
177       final <- glm(Y~.,data=data.frame(Y=data_train$Y,X=data_train$X),
              family="binomial")
178       betahat <- as.numeric(final$coefficients[-1])
179     }
180
181     ###############
182     ### Predict ###
183     ###############
184     if(regularization=="no"){
185       p <- predict(final,data.frame(X=data_test$X),type="response")
186     }else if(regularization=="scad"){
```

```
187       p <- predict(final,data_test$X,type="response")
188     }else if(regularization=="alasso"){
189       p <- predict(cvresult,s=optimal_lambda,newx=data_test$X,type="
              response")
190         }else{
191       p <- predict(cvresult,s="lambda.min",newx=data_test$X,type="
              response")
192     }
193
194     #Return relevant information
195     c( sum(betahat!=0)
196        ,mean(betahat[beta==0] ==0) #TPR for beta
197        ,mean(betahat[beta!=0] !=0) #FPR for beta
198        ,mean(1*(p[data_test$Y==1]>=0.5)) #TPR for predicitons
199        ,mean(1*(p[data_test$Y==0]<0.5)) #TNR for predicitons
200        ,package::AUC(data_test$Y,p)$AUC
201        ,package::BS(data_test$Y,p) #Brier score
202        ,hyper_param)
203   }
204   stopCluster(cl)
205   #return result
206   parallel_result
207 }
208
209 #Number of simulations
210 m <- 200
211 #Number of cores to be used
212 workers <- 8
213
214 regularization <- c("no","lasso","ridge","scad","elnet","alasso")
215 simulation <- 1:7
216
217 #Run simulations
218 for(i in simulation){
219         for(j in 1:length(regularization)){
220           result <- 0
221                 result <- study1(m=m,regularization=regularization[j],
                          cores=workers,simulation=simulation[i])
222         #Write result to file
223 write.table(result,paste(regularization[j],simulation[i],sep="_"),row.
      names=F,col.names=F)
224                 }
225 }
```

A.4 Chapter 7

The following R script computes the AUC for logistic regression for lasso, ridge, adaptive lasso and elastic net.

```
glmnet_auc <- function(X_train,Y_train,X_test,Y_test,alpha,weights,
    lambda_sequence){
  #Train model
  cvresult <- glmnet(X_train,Y_train,family="binomial",alpha=alpha,
      lambda=lambda_sequence,penalty.factor=weights)
  #Predict the test data, this produces a matrix
  predicted <- predict(cvresult,newx=X_test,type="response")
  AUC_v <- 1:length(lambda_sequence)*0
  #Loop over all predicted values, one iteration for each lambda value
  if(ncol(predicted)==length(lambda_sequence)){
    for(i in 1:length(lambda_sequence)){
      Y_pred <- predicted[,i]
      AUC_v[i] <- AUC(Y_test,Y_pred)$AUC
    }
  }
  return(AUC_v)
}
```

The following R script takes the re-sampled training data and uses these in the cross-validation prodecure as described in Chapter 7. Synthetic observations are labeled by $Y = -1$.

```
cv_glmnet <- function(X,Y,folds,re_method="no",re_balance="0",alpha,
    weights_gamma=-1,lambda_sequence){
  n <- nrow(X)
  k <- length(unique(folds))

  #initialize
  cv_AUC <- matrix(0,nrow=k,ncol=length(lambda_sequence))

  #for each k
  for(i in 1:k){

    #The following if-tests are necessary in order to treat the over-
        sampled data correctly in the cross-validaiton procedure.
    #See comments regarding this in Chapter 7.

    #If under-sampling is used
    if(re_method == "under"){

      ind_train <- which((folds!=i) & (Y != -1))
      ind_test <- which(folds==i)

      Y_train <- Y[ind_train]
```

```
21      X_train = X[ind_train,]
22      X_test = X[ind_test,]
23      Y_test = Y[ind_test]
24      Y_test[Y_test==-1] <- 0
25
26
27      #If weights should be applied, i.e. adaptive lasso
28      if(weights_gamma != -1){
29        ind_train_w <- which((folds!=i))
30        X_train_w <- X[ind_train_w,]
31        Y_train_w <- Y[ind_train_w]
32
33        Y_train_w[Y_train_w==-1] <- 0
34
35        folds_w <- stratified(X_train_w,Y_train_w,k=5)
36
37        weight_cvresult <- cv.glmnet(X_train_w,Y_train_w,family="
              binomial",alpha=0,type.measure="auc",foldid=folds_w,nlambda
              =50)
38        beta_values <- as.numeric(coef(weight_cvresult,s="lambda.min")
              )[-1]
39
40        weights <- abs(1/beta_values)
41        weights[weights==Inf] <- 999999
42        weights <- weights^weights_gamma
43      }else{
44        weights <- rep(1,ncol(X_train))
45      }
46    #If random over-sampling or SMOTE is used
47    }else if(re_method == "over" || re_method == "smote"){
48
49      ind_train <- which(folds!=i)
50      ind_test <- which(folds==i & (Y != -1))
51
52      Y_train <- Y[ind_train]
53      X_train = X[ind_train,]
54      X_test = X[ind_test,]
55      Y_test = Y[ind_test]
56
57      Y_train[Y_train==-1] <- 1
58
59      #If weights should be applied, i.e. adaptive lasso
60      if(weights_gamma != -1){
61        ind_train_w <- which((folds!=i) & (Y != -1))
62        X_train_w <- X[ind_train_w,]
63        Y_train_w <- Y[ind_train_w]
64
65        folds_w <- stratified(X_train_w,Y_train_w,k=5)
66
```

```
67        weight_cvresult <- cv.glmnet(X_train_w,Y_train_w,family="
              binomial",alpha=0,type.measure="auc",foldid=folds_w,nlambda
              =50)
68        beta_values <- as.numeric(coef(weight_cvresult,s="lambda.min")
              )[-1]
69
70        weights <- abs(1/beta_values)
71        weights[weights==Inf] <- 999999
72        weights <- weights^weights_gamma
73      }else{
74        weights <- rep(1,ncol(X_train))
75      }
76    #If no re-sampling
77    }else{
78
79      ind_train <- which(folds!=i)
80      ind_test <- which(folds==i)
81
82      Y_train <- Y[ind_train]
83      X_train = X[ind_train,]
84      X_test = X[ind_test,]
85      Y_test = Y[ind_test]
86
87      #If weights should be applied, i.e. adaptive lasso
88      if(weights_gamma != -1){
89        ind_train_w <- which(folds!=i)
90        X_train_w <- X[ind_train_w,]
91        Y_train_w <- Y[ind_train_w]
92
93        folds_w <- stratified(X_train_w,Y_train_w,k=5)
94
95        weight_cvresult <- cv.glmnet(X_train_w,Y_train_w,family="
              binomial",alpha=0,type.measure="auc",foldid=folds_w,nlambda
              =50)
96        beta_values <- as.numeric(coef(weight_cvresult,s="lambda.min")
              )[-1]
97
98        weights <- abs(1/beta_values)
99        weights[weights==Inf] <- 999999
100       weights <- weights^weights_gamma
101     }else{
102       weights <- rep(1,ncol(X_train))
103     }
104   }
105
106   #Run the function "glmnet_auc" for the current training and test
          data
107   cv_AUC[i,] <- glmnet_auc(X_train = X_train,
108                            Y_train = Y_train,
```

```
109                             X_test = X_test,
110                             Y_test = Y_test,
111                             alpha=alpha,
112                             weights=weights,
113                             lambda_sequence = lambda_sequence)
114   }
115   return(list(cv_AUC=cv_AUC,lambda_sequence=lambda_sequence))
116 }
```

The R script for running simulations in Chapter 7 is very similar to that given for Chapter 6 above. The difference is that the training data are re-sampled prior to model training.

Code for Chapter 8 has not been provided. This was done to conserve space. Additionally, the analyses have been described thoroughly in the text of Chapter 8.

Bibliography

Balakrishnan, N. (1991). *Handbook of the logistic distribution*. CRC Press.

Berset, A., S. Hussain, and P. A. Paulsen (2016). "Prediktiv modell har økt treffprosenten på oppgavekontroll". In: *Skatteetatens Analysenytt 1/2016*, pp. 16–19.

Bolton, R. J. and D. J. Hand (2002). "Statistical fraud detection: a review". In: *Statistical Science* 17.3, pp. 235–249.

Bølviken, E. (2014). *Computation and Modelling in Insurance and Finance*. Cambridge University Press.

Breiman, L. (1996a). "Bagging predictors". In: *Machine Learning* 24.2, pp. 123–140.

Breiman, L. (1996b). "Heuristics of instability and stabilization in model selection". In: *The Annals of Statistics* 24.6, pp. 2350–2383.

Breiman, L. (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32.

Brier, G. W. (1950). "Verification of forecasts expressed in terms of probability". In: *Monthey Weather Review* 78.1, pp. 1–3.

Chan, P. K. et al. (1999). "Distributed data mining in credit card fraud detection". In: *IEEE Intelligent Systems and Their Applications* 14.6, pp. 67–74.

Chawla, N. V., N. Japkowicz, and A. Kotcz (2004). "Special issue on learning from imbalanced data sets". In: *The Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining Explorations* 6.1, pp. 1–6.

Chawla, N. V. et al. (2002). "SMOTE: synthetic minority over-sampling technique". In: *Journal of Artificial Intelligence Research* 16, pp. 321–357.

Fan, J. and R. Li (2001). "Variable selection via nonconcave penalized likelihood and its oracle properties". In: *Journal of the American Statistical Association* 96.456, pp. 1348–1360.

Fawcett, T. (2006). "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8, pp. 861–874.

Friedman, J., T. Hastie, and R. Tibshirani (2010). "Regularization paths for generalized linear models via coordinate descent". In: *Journal of Statistical Software* 33.1, p. 1.

Genest, C. and A.-C. Favre (2007). "Everything you always wanted to know about copula modeling but were afraid to ask". In: *Journal of Hydrologic Engineering* 12.4, pp. 347–368.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer New York.

Japkowicz, N. (2000). "The Class Imbalance Problem: Significance and Strategies". In: *In Proceedings of the 2000 International Conference on Artificial Intelligence*, pp. 111–117.

Jong, P. de and G. Z. Heller (2008). *Generalized Linear Models for Insurance Data*. Cambridge University Press.

Kim, J. H. (2009). "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap". In: *Computational Statistics & Data Analysis* 53.11, pp. 3735–3745.

Kohavi, R. (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *International Joint Conference on Articial Intelligence* 14.2, pp. 1137–1145.

Kubat, M. and S. Matwin (1997). "Addressing the curse of imbalanced training sets: one-sided selection". In: *International Conference on Machine Learning*. Vol. 97, pp. 179–186.

Le Cessie, S. and J. C. Van Houwelingen (1992). "Ridge estimators in logistic regression". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 41.1, pp. 191–201.

Leeb, H. and B. M. Pötscher (2008). "Sparse estimators and the oracle property, or the return of Hodges' estimator". In: *Journal of Econometrics* 142.1, pp. 201–211.

Ling, C. X. and C. Li (1998). "Data mining for direct marketing: Problems and solutions." In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 73–79.

Løland, A., A. Berset, and I. Hobæk Haff (2017). "Er maskinlæring framtida i Skatteetaten?" In: *Praktisk økonomi & finans* 33.3, pp. 344–352.

Oommen, T., L. G. Baise, and R. M. Vogel (2011). "Sampling bias and class imbalance in maximum-likelihood logistic regression". In: *Mathematical Geosciences* 43.1, pp. 99–120.

Santos, M. S. et al. (2018). "Cross-Validation for Imbalanced Datasets: Avoiding Overoptimistic and Overfitting Approaches". In: *IEEE Computational Intelligence Magazine* 13.4, pp. 59–76.

Sklar, M. (1959). "Fonctions de repartition an dimensions et leurs marges". In: *Publications de l'Institut Statistique de l'Université de Paris* 8, pp. 229–231.

Solberg, A. S. and R. Solberg (1996). "A large-scale evaluation of features for automatic detection of oil spills in ERS SAR images". In: *International Geoscience and Remote Sensing Symposium*. Vol. 3, pp. 1484–1486.

Swets, J. A. (1988). "Measuring the accuracy of diagnostic systems". In: *Science* 240.4857, pp. 1285–1293.

Tibshirani, R. (1996). "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1, pp. 267–288.

Tutz, G. and H. Binder (2006). "Generalized additive modeling with implicit variable selection by likelihood-based boosting". In: *Biometrics* 62.4, pp. 961–971.

Van Hulse, J., T. M. Khoshgoftaar, and A. Napolitano (2007). "Experimental Perspectives on Learning from Imbalanced Data". In: *Proceedings of the 24th International Conference on Machine Learning*, pp. 935–942.

Zou, H. (2006). "The adaptive lasso and its oracle properties". In: *Journal of the American Statistical Association* 101.476, pp. 1418–1429.

Zou, H. and T. Hastie (2005). "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2, pp. 301–320.