MACHINE LEARNING AND ITS APPLICATIONS WITHIN INSURANCE HIT RATES AND CREDIT RISK MODELLING

LINUS BLOMGREN AND HAMPUS VITESTAM

Master's thesis
2017:E23

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Abstract

This thesis aims to shine light on some different machine learning methods. As reference a more common statistical prediction method, namely the generalized linear model, is applied to compare the results of the machine learning methods. Six different machine learning methods are investigated. These methods are explained in detail and used to predict hit rates within insurance customers. To further explore the data sets and the methods, the data sets are rebalanced to deal with skewness of the target class. The insurance data set used contains 86 features, including the target feature, which can be troublesome in some cases, and therefore a feature reduction analysis is performed. Further the positives and negatives of the different methods and how to put machine learning in practice was discussed. Lastly a new data set is introduced and the machine learning methods are used to assess the risk of default within credit customers.

The results show that random forest perform best of the different data sets, and it is fairly easy to interpret. The k-nn, naïve Bayes and decision tree do not perform as well as the random forest but are easier to use and requires much less computing time to tune and train. These less computational complex methods can be good when much data is available, but is inferior to regression methods when that is not the case. The support vector machine and the neural network are complex but have potential for greatness. Further investigation into the different models we used are needed, especially the support vector machine and the neural network.

Key words: Machine learning, Artificial intelligence, Insurance, Credit risk, SMOTE, k-NN, Naïve Bayes, Decision tree, Random forest, Support vector machine, Neural network, Generalized linear model, Receiver operating characteristics, Hit rate

Contents

Chapter 1

Introduction

Machine learning is defined as a set of methods which can be used to automatically detect patterns in data. The uncovered patterns are usually used for predicting future data. Machine learning originates in the field of computer science as it aims to provide automated methods for data analysis and decision making. However these kinds of problems come with a lot of uncertainty. Machine learning is also closely related to the field of statistics to deal with the issue of uncertainty. It originates over 50 years ago but has in the recent decades become more popular and put in practice. The last decade there was a great discovery within neural networks and since then applications of machine learning are increasing within all fields of computer science.

There are mainly two types of machine learning: supervised and unsupervised learning. In supervised learning, or predictive learning, the target output is known, i.e. the aim is to map inputs $\boldsymbol{x}$ to outputs $\boldsymbol{y}$ given a data set of input-output pairs. The data set is called training set as it is used for training the models. Usually a part of the data set is put aside and used for validating the model's performance, see Section 2.1. The input parameters are commonly referred to as features or attributes, in this report they will be referred to as features, while the output can be either categorical or real-valued resulting in classification or regression, respectively. Supervised learning is the most common type of machine learning as you often know what it is that you want to predict. It is therefore the applied type in this report.

As for the second type, unsupervised learning, or descriptive learning, the data has no target output, instead the goal is to identify patterns in the data that might be interesting. Problems that can be solved using unsupervised learning may be to discover similarities within groups of the data or to determine the distribution of the data within the input space. Unsupervised learning is a less well-defined problem compared to supervised learning since it is unknown what to look for and since there is no apparent performance measure.

A third type of machine learning, which has gained more attention in recent years, is known as reinforcement learning. Reinforcement learning is concerned with finding actions suitable to the environment in which it operates by maximizing some notion of reward. Unlike supervised learning, reinforcement learning algorithms are not given examples of optimal output but must instead discover them for themselves by trial and error, much like a baby learning to walk. The algorithm's interaction with the environment is typically described by sequences of states and actions that will be differently rewarded depending on the outcome.

This report will first introduce the theory around different machine learning methods. Then it will take on customer relations within insurance companies, focusing mainly on predicting hit rates for caravan insurances. Two attempts to improve the results are carried out: firstly by rebalancing the data and secondly by reducing the amount of features in the data. This report also includes a comparison of how the machine learning methods perform on a credit data set to predict defaults.

1.1 Objective

The objective of this report is to give an overview of some machine learning methods. It aims to explain the idea and theory behind each chosen method in a way so that little or no previous knowledge about machine learning is needed. As machine learning methods are currently not commonly used methods within the field of finance. The methods will be evaluated on two typical financial data sets: insurance and credit risk, to examine the methods' applicability within the field. The performance of each method will be compared to each other, but also to a benchmark model, the generalized linear model, that have been trained on the same data. This is done to compare to a statistical model that are commonly used on similar problems.

1.2 Limitations

To narrow down our scope we have chosen six machine learning methods. They are naïve Bayes, k-nearest neighbors, decision tree, random forest, support vector machine and neural network. We chose them as they have different approaches, and they are good representatives for the different available machine learning methods. The support vector machine was tuned and trained using three different kernel functions. They are the linear, polynomial and radial basis kernel. These three functions are the most commonly used kernel functions since they cover many different invariances within the data.

Chapter 2

Theory

This chapter presents the theory behind the machine learning methods applied and the performance measurement used. The example pictures in this chapter are generated from the famous iris data set collected by Anderson (1936) and first analyzed by Fisher (1936). The data set consists of 4 features on each of 50 data points from each of three species of iris: *Iris setosa*, *versicolor* and *virginica*. The features sepal length and width and petal length and width, respectively, are given in centimeters. The data was obtained through the programming language R by R Core Team (2016) in which it is built-in.

2.1 Overfitting and cross validation

One of the risks when training a model to fit the training data is overfitting. This means that the model describes the noise rather than captures the underlying relationship or trend of the points in the data set. In other words, the model will fit the training data too well and overreacts to any minor fluctuation in the data used to fit the model. The model's predictive performance will therefore be very far off when introduced to a new data set. Overfitting of a model is typically due to the model being excessively complex for the data. There are different ways to deal with the overfitting problem such as reducing the dimensionality of the input parameters to reduce the risk of the model being too complex.

Another way of reducing the overfitting problem is to use cross validation when training the model. In cross validation the training data is split into K subsets, called folds. For $k \in 1, ..., K$ a model is trained on every fold except the k'th and its performance validated on the k'th fold. This is repeated for all K folds and the averaged performance over all folds is used as the performance score for the model (Murphy, 2012, p. 24). Another advantage of cross validation is that the technique solves the problem of having limited data since it uses all available data to asses performance. The repeated training is however the technique's biggest drawback as this process itself becomes computationally expensive due to the training process being carried out K times. (Bishop, 2006, pp. 32-33)

2.2 Weak and strong learners

Learner is a collective term for algorithms used for prediction, classification, etc. A learner's performance can be indicated as either *random*, *weak* or *strong.* A random learner's performance is equal to guessing the outcome without taking any features into consideration. That is, a random learner on a binary classification problem has a performance of 0.5 as there is a 50% chance of correct classification by just guessing. A weak learner performs slightly better than a random learner, but its performance is still relatively poor. The closer to 1 (100% correctly classified) a learner's performance is, the stronger it is. It is often, but not always, the case that a weak learner is also computationally simple while strong learners can be computationally expensive.

2.3 Ensemble Learning

Within machine learning the term ensemble learning corresponds to the methods that combine multiple predictions to form a (hopefully) better predictor. It can be describes as searching trough a space of predictors to find the one that has a suitable predictive performance for the particular problem. Even if there is a very suitable predictor for the particular problem, it can be very difficult to find it in the predictor space. The idea is to combine several predictors in order to add certainty for new predictions. It can be seen as performing a lot of extra computations to compensate for poor learning algorithms. With y as the output and $\mathbf{x}$ as the input vector, the general idea of the method is to construct a learner, $f(y|\mathbf{x}$, on the form of

$$f(y|\mathbf{x}) = \sum_{m \in \mathcal{M}} w_m f_m(y|\mathbf{x}),$$

where the w_m are tunable parameters, $\mathcal{M}$ is the predictor space and f_m are predictors in said space (Murphy, 2012, p. 580). The chosen learners in the prediction space are usually weak learners for computational reasons, but the resulting learner f is hopefully a strong learner. The learning algorithms f_m are usually generated from the same type of weak learner, but this can be extended to multiple types of weak learning algorithms.

2.3.1 Boosting

Boosting is an algorithm that aims to create a strong learner from a set of weak learners. The method consists of iteratively adding weak learners applied sequentially to different versions of the data. Each version of the data is weighted differently so that more weight is given to data points that were more likely to be misclassified by the previous weak learner. The sequence of weak learners are then weighted together to form a strong learner. It has been shown that the performance of any weak learner, that performs slightly better than chance, can be boosted arbitrarily high. (Murphy, 2012, pp. 554-555)

The method works especially well for classifiers. There are many boosting algorithms, and the main difference between them is the method of weighing the data points in each iteration of the sequence.

2.3.2 Bagging

Bagging, or bootstrap aggregating as the acronym stands for, is a sampling method commonly applied to machine learning techniques. The method aims to improve the accuracy of the machine learning algorithm and to reduce the variance as well as the risk of overfitting. The bagging method generates new training sets $\mathcal{D}_i$ by sampling uniformly with replacement from the original training data set $\mathcal{D}$. For each training set a predictor is calculated by:

$$f(x) = \sum_{m=1}^{M} \frac{1}{M} f_m(x), \tag{2.1}$$

where $f_m(x)$ is the weak predictor model. The output is the aggregated average over all the versions of predictors when predicting a numerical outcome. When predicting a class, the majority of votes is instead used. (Murphy, 2012, p. 551)

Since bagging is an additive ensemble technique, adding multiple linear models will result in a new linear model. By adding randomness the mean-squared error produced by bagging will be less than or equal to the mean-squared error of the original predictor. Breiman (1996) shows both theoretically and empirically that the bagging method works well in improving the performance of unstable procedures, such as decision trees. However there is a limit to how much improvement can be achieved by bagging. A predictor that performs at the limit of accuracy attainable on a data set cannot be improved further, no matter how much bagging is done.

2.4 Naïve Bayes

Naive Bayes is a classification method that uses the simplification that all features are independent. The model uses the training set to calculate the probabilities of being in a certain class given all features one at a time. When introducing new points to the model, the joint probability of all its features is calculated. Since these are all assumed to be independent, Bayes' formula and the chain rule can be applied and the following relation holds:

$$p(C_k|x_1, x_2, ..., x_{n-1}, x_n) = \frac{p(x_1, x_2, ..., x_{n-1}, x_n, C_k)}{p(\boldsymbol{x})} = \frac{p(C_k)}{p(\boldsymbol{x})} \prod_{i=1}^{n} p(x_i|C_k),$$

where x_i represent the n different features and C_k represents the k different classes. The method is then to find the class which has the highest probability and classify the new data point as this class. Since the class does not depend on $p(\boldsymbol{x})$ this is just a scaling factor and our problem to be solved is (Lewis, 1998):

$$\hat{y} = \underset{k \in \{1,...,K\}}{\operatorname{argmax}} \, p(C_k) \prod_{i=1}^{n} p(x_i|C_k).$$

When the features' values are continuous the distribution of the features of the different classes has to be calculated. There are several distributions that can be fitted to this but it is common to use a Gaussian distribution and to calculate the probability of this. (Murphy, 2012)

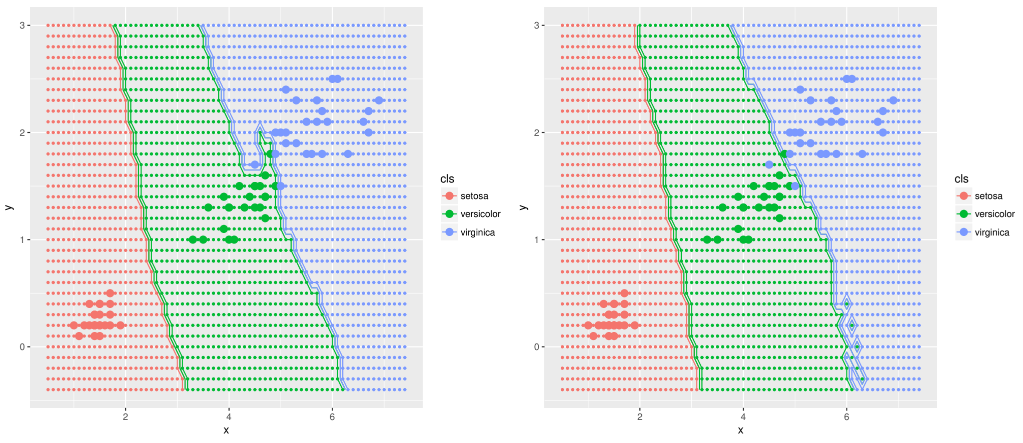
2.5 k-Nearest Neighbors

The method called k-Nearest Neighbors (k-NN) is one of the simplest machine learning methods. It uses a new data point to find the k nearest neighbors of this point within the training set. These points can then be used in the classification of the new point based on the classification of these k nearest neighbors. The method uses the majority vote of these neighbors to decide what class the new point is classified as. The algorithm can also be used for regression. It uses the k nearest neighbors to find the value of the new point by computing the average of these values. To improve the predictions, weights are added to the algorithm. By using the inverse relative distance and averaging over these the class or value is decided. The following formula is used for regression:

$$\hat{y} = \frac{1}{\sum_{i=1}^{k} e^{-D_i}} \sum_{i=1}^{k} e^{-D_i} y_i$$

where y_i is the value of the training point, D_i is the distance to the new point and $\hat{y}$ is the value of the new point. When weights are used for classification the sum of the inverse relative distances of each class is compared to each other and the class maximum value is chosen. The distance in this formula can be calculated in different ways but a common way is to use the Euclidean distance. The next thing to do is to choose the best number of neighbors. The model is then optimized by using the least squared error when using regression and by minimizing the misclassification error when using classification. (Hill and Lewicki, 2007)

In Figure 2.1 the k-NN approach has been used for classification on the iris data set for k = 1 and k = 9. The training data points are marked in the plot and a new observation would be classified according to the field in which it would be placed, either red, green or blue.



(a) K-nearest neighbors using k=1. (b) K-nearest neighbors using k=9.

Figure 2.1: Two figures of k-NN classification on the same data set with two different k-values.

2.6 Decision tree

The decision tree is a model where you partition the input space into cuboid regions with edges aligned with the axes. This is done with a binary tree where the data set is divided by a threshold in one feature of the data set. This is called a node. The data set is split into two or more branches, and this is repeated a finite number of times and end up in terminal nodes that are called leaves.

To build the tree, nodes have to be chosen. Exploring all possible combination of nodes is often not feasible because there can be an unlimited amount of possible splittings. Therefore an algorithm is used to maximize the information gain by minimizing the uncertainty of the input spaces using the following equation:

$$\text{Costgain}(\mathcal{D}) = \text{Cost}(\mathcal{D}) - \sum_{i=1}^{n} \frac{|\mathcal{D}_i|}{|\mathcal{D}|}\text{Cost}(\mathcal{D}_i)$$

where $\mathcal{D}$ is the data set before splitting the node and $\mathcal{D}_i$ is the data set of the i'th branch of the node after the split (Murphy, 2012, pp. 545-548). For classification this cost function can be represented by three different functions: misclassification rate, entropy or Gini index. The misclassification cost is described as follows:

$$\text{Cost}(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{i\in\mathcal{D}} \mathbb{I}(y_i \neq \hat{y}_c),$$

where $\hat{y}_c$ is the most probable class defined as $\hat{y}_c = \underset{c}{\text{argmax}}(\hat{\pi}_c)$ and y_i is the outcome of the i'th point in the data set. Here $\hat{\pi}_c$ is the probability that a data point in the region will be classified in a certain class. This is formulated as follows:

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|}\sum_{i\in\mathcal{D}} \mathbb{I}(y_i = c).$$

The cost function of entropy is explained as follows:

$$\text{Cost}(\mathcal{D}) = -\sum_{c=1}^{C} \hat{\pi}_c \log_2(\hat{\pi}_c).$$

The Gini index-cost function can be calculated with the following formula:

$$\text{Cost}(\mathcal{D}) = \sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c).$$

For regression the split is chosen by choosing the feature that minimizes the sum of squares:

$$\text{Cost}(\mathcal{D}) = \sum_{i=1}^{k} (y_i - t)^2.$$

where t is the threshold for the split. The threshold is chosen to minimize the sum of squares, and the optimal choice of threshold is the mean of y:

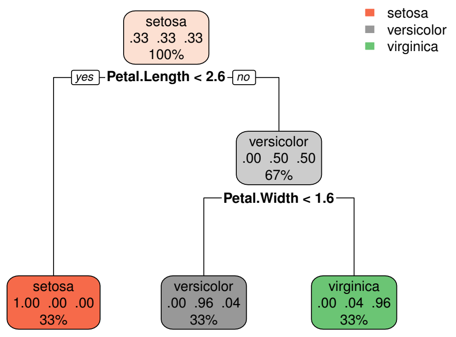
$$t = \bar{y} = \frac{1}{|\mathcal{D}|}\sum_{i\in\mathcal{D}} y_i.$$

The best node is chosen repeatedly until the tree is fully grown. The theory of deciding when a tree is fully grown and how to optimize the tree is explained in the next section.

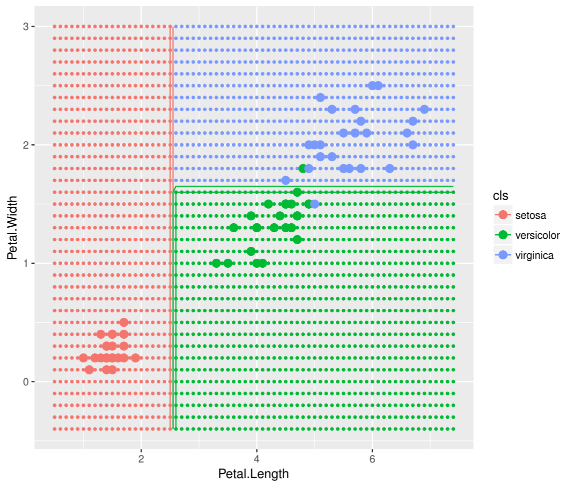
2.6.1 Overfitting – Stopping criterion – Pruning

When using the training set, the optimal tree for this set exists and is obtained when the data set is completely separated. This does not necessarily reflect the truth but is rather a symptom of overfitting. Another way to decide how far to build the tree is to define a stopping criterion. This can be a so called complexity criterion, a maximum number of nodes or a minimum number of elements in the node. The complexity criterion is based on a complexity parameter, which is a measurement of how much information you can gain by splitting after this point. These methods can cut computational costs, but they are not perfect.

One method that can then be applied is to build a large tree and then "prune" the tree to find the optimal tree. The different subtrees of the original large tree are checked using either cross validation or a test set and then the sub-tree with the least error is chosen.



(a) A decision tree with three terminal nodes.



(b) The decision trees predictions.

Figure 2.2: A pruned decision tree on the iris data set and its predictions.

In the left picture of the Figure 2.2, a decision tree has been grown from the iris data set with only three terminal nodes. In the right picture the trees corresponding to the prediction of new data points are shown, with the training data points marked.

2.6.2 Random forest

The random forest method is a modification of bagging that grows multiple independent decision trees and averages them together. In equation (2.1) this means that $f_m(x)$ is the m'th tree. (Murphy, 2012, pp. 550-551)

Even though bagging will result in reduced variance, the variance reduction has a limit since rerunning the same learning algorithms on different subsets will lead to some highly correlated predictors. Random forests try to decorrelate the trees by not only selecting a random subset of the input data but also by randomly selecting a subset of the input features. For a data set with p input features, $m \leq p$ is randomly chosen for each split in the tree. For classification problems it is recommended to use $m = \sqrt{p}$ while for regression problems the recommendation is $m = p/3$ (Hastie et al., 2009, p. 592). The prediction accuracy of random forests is often very good, however the easy interpretation that decision trees provide is lost due to the multiple trees.

2.7 Support Vector Machines

When talking about Support Vector Machines (SVMs), it is common to talk about the two-class classification problem, but it can also be used for multiclass classification and regression. The method is to train a hyperplane in a feature space that splits the data set according to the classifications. This hyperplane is optimized to maximize the perpendicular distance to the closest data points. The model will then only depend on the closest data points, that are called support vectors, hence it is independent of all other data points.
Figure 2.3 shows a SVM of the iris data set. The SVM is created using the two features petal length and petal width. The line between the blue area and the white area is a hyperplane that completely separates one, class and the line between the white area and the pink area is a hyperplane that separates two classes using a soft margin. The points marked with an X are the support vectors. When considering the two-class classification, the following linear model is used:

$$y(x) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b,$$

where $\boldsymbol{w}$ are the data weights, $\phi(\boldsymbol{x})$ denotes the fixed feature-space transformation and b is a bias parameter. The target values $t_1, ..., t_N$ corresponding to the N training vectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ are labelled with output values $t_n = \{-1, 1\}$ for computational convenience, which makes the new data points to be classified as the sign of $y(\boldsymbol{x})$. If the data points are linearly separable in the feature space, the purpose of an SVM is to minimize the generalization error. This is done by maximizing the margin,

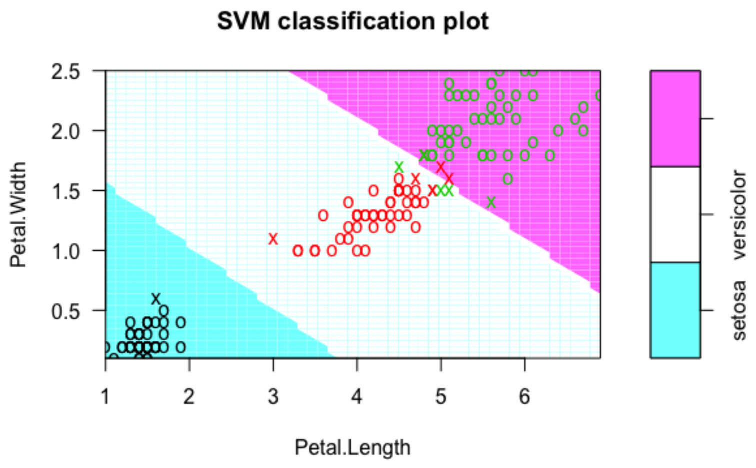


Figure 2.3: A plot of a three class classification SVM-model.

which is the distance between the decision boundary and the closest samples. The samples at the margin are called support vectors, hence the name support vector machine. The distance of a point $\boldsymbol{x}_n$ to the hyperplane where $y(\boldsymbol{x}) = 0$ is given by $|y(\boldsymbol{x}_n)|/||\boldsymbol{w}||$, where $||\boldsymbol{w}||$ is the euclidean norm. Since $t_n y(\boldsymbol{x}_n) > 0$, the following equation gives the perpendicular distance of point $\boldsymbol{x}_n$ to the decision surface:

$$\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||} = \frac{t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b)}{||\boldsymbol{w}||}. \tag{2.2}$$

The problem is then to maximize equation (2.2) according to the point with the minimum perpendicular distance to the decision surface. This is done by solving the following equation:

$$\underset{\boldsymbol{w},b}{\operatorname{argmax}} \left\{ \frac{1}{||\boldsymbol{w}||} \min_n [t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b)] \right\}.$$

But by multiplying $\boldsymbol{w}$ and b with the same constant will not change equation (2.2). The following equation can be used for the points closest to the decision surface:

$$t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b) = 1.$$

This means that the following equation is a constraint for any data point in the data set:

$$t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b) \geq 1. \tag{2.3}$$

The constraints where the equality holds are called active constraints and the inequality constraints are called inactive constraints. There will always be a point closest to the decision surface, and when the margin is maximized there will always be at least two. The problem is to maximize $||\boldsymbol{w}||^{-1}$ subject to the constraints in (2.3). For mathematical convenience it will be transformed. Using that this is equivalent to minimizing $||\boldsymbol{w}||$ or $||\boldsymbol{w}||^2$ and multiplying by a constant, the problem will have the same solution as the following equation:

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2,$$

subject to the constraints in equation (2.3). The bias parameter b is taken out of this equation but is implicitly accounted for in the constraints. Therefore this problem can be solved by introducing the Lagrangian function:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{n=1}^{N} a_n\{t_n(\boldsymbol{w}^T\phi(x_n) + b) - 1\}, \tag{2.4}$$

where $\boldsymbol{a} = (a_1, ..., a_n)^T, a_n \geq 0$ are the Lagrange multipliers. The minus sign in (2.4) is there because it is being minimized with respect to $\boldsymbol{w}$ and b and maximized with respect to $\boldsymbol{a}$. Using the derivatives of (2.4) with respect to $\boldsymbol{w}$ and that b equal to zero the following equations are obtained:

$$\boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \phi(\boldsymbol{x})$$

$$0 = \sum_{n=1}^{N} a_n t_n.$$

These results are used to eliminate $\boldsymbol{w}$ and b from (2.4) and the results are

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m), \tag{2.5}$$

constrained to

$$a_n \geq 0, \quad n = 1, ..., N$$

$$\sum_{n=1}^{N} a_n t_n = 0.$$

To avoid working explicitly in the feature space, the kernel function is introduced:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})\phi(\boldsymbol{x}').$$

If this kernel is positive definite, it will make sure that equation (2.5) is bounded from below and the problem is a well-defined optimization problem. For the purpose of classification, the sign of $y(\boldsymbol{x})$ is expressed in terms of the kernel and $\boldsymbol{a}$ as follows:

$$y(\boldsymbol{x}) = \sum_{n+1}^{N} a_n t_n k(\boldsymbol{x}_n, \boldsymbol{x}_m) + b, \tag{2.6}$$

This constrained optimization problem satisfies the Karush-Kuhn-Tucker (KKT) conditions, which means that the following properties hold:

$$a_n \geq 0$$

$$t_n y(\boldsymbol{x}_n) - 1 \geq 0$$

$$a_n\{t_n y(\boldsymbol{x}_n) - 1\} = 0.$$

These constraints say that if $a_n > 0$, then $t_n y(\boldsymbol{x}_n) = 1$, which means that only the support vectors have a Lagrange multiplier different from zero. Therefore only the active constraints will be considered in the sum in equation (2.6). The other points in the training set are discarded, and the solution does not depend on them. When the values of $\boldsymbol{a}$ are found, the value of the bias parameter b is found by noting that any support vector $\boldsymbol{x}_n$ satisfies $t_n y(\boldsymbol{x}) = 1$ and using (2.6). This creates the following equation:

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) + b \right) = 1, \tag{2.7}$$

where $\mathcal{S}$ denotes the indices of the support vectors. The solution can be found by using any of the support vectors, but a better method is to multiply both sides of equation (2.7) by t_n, use the fact that $t_n^2 = 1$ and then take an average using all support vectors. The solution then becomes (Bishop, 2006, 330):

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) \right).$$

When linearly separating the classes according to the feature space $\phi(\boldsymbol{x})$ the solution does not have to represent the best generalization. There is a possibility that the class distributions overlap in the problem at hand. Therefore a modified version of the SVM is proposed, where a slack variable $\xi_n \geq 0$ is introduced for when data points are on the wrong side of the margin hyperplane. The slack variable is a linear penalty, defined as the distance between the point and the margin, $\xi = |t_n - y(\boldsymbol{x}_n)|$. This is called giving the problem a soft margin and allows some points to be misclassified. The exact classification constraint is then defined as:

$$t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n, \quad n = 1, ..., N \tag{2.8}$$

which has the same properties as $t_n y(\boldsymbol{x}_n) \geq 0$ because ξ is 1 in the decision surface. The optimization problem is then to minimize the following equation:

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||\boldsymbol{w}||^2, \tag{2.9}$$

where $C > 0$ is a parameter that decides how hard the optimization is penalized by the slack variables. It is true that if $C \to \infty$, the case where there is no room for misclassification is obtained. To minimize (2.9) under the constraints in (2.8) the following Lagrangian is formed:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n \{t_n y(\boldsymbol{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n, \tag{2.10}$$

where $a_n \geq 0$ and $\mu_n \geq 0$ are the Lagrange multipliers. The KKT conditions are then as follows:

$$a_n \geq 0$$

$$t_n y(\boldsymbol{x}_n) - 1 + \xi_n \geq 0$$

$$a_n(t_n y(\boldsymbol{x}_n) - 1 + \xi_n) = 0$$

$$\mu_n \geq 0$$

$$\xi_n \geq 0$$

$$\mu_n \xi_n = 0,$$

where $n = 1, ..., N$. The Lagrangian (2.10) is then optimized according to $\boldsymbol{w}$, b and ξ_n which gives the following results:

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \phi(\boldsymbol{x}_n),$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} a_n t_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = 0 \Rightarrow a_n = C - \mu_n$$

and from this the following dual Lagrangian is as follows:

$$\tilde{L}(\boldsymbol{a}) = \sum_{n+1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m). \tag{2.11}$$

This is the same as in the non-misclassification case except that the constrains differ. The Lagrangian (2.11) is therefore optimized with respect to $\boldsymbol{a}$, subject to the following constraints:

$$0 \leq a_n \leq C$$

$$\sum_{n=1}^{N} a_n t_n = 0,$$

for $n = 1, ..., N$. The predictions for new data are still made by using (2.6) but with the new Lagrangian multipliers $\boldsymbol{a}$. The constraints work similarly here as in the non-misclassification case such that if the following equality hold:

$$t_n y(\boldsymbol{x}_n) = 1 - \xi_n,$$

then a_n differ from zero. The constraints also require that if $a_n < C$, that $\mu = 0$, which implies that $\xi = 0$ and therefore that the point is located on the margin. If $a_n = C$, then the point is on the other side of the margin, and if $\xi > 1$ it is misclassified. To determine the bias parameter b, the same equation as in the non-misclassification case is used:

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) \right) = 1.$$

This time only the points where $\xi_n = 0$ can be used because then $t_n y(\boldsymbol{x}_n) = 1$. The solution of b is then:

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) \right),$$

where $\mathcal{M}$ denotes the indices of data points where $\xi = 0$. (Bishop, 2006, pp.326-345)

2.7.1 Choice of Kernel

The kernel function takes the data set into a feature space according to the properties of the function. This is useful when the problem is not linear and can therefore be better solved in another space. The kernel function can be many different functions but the polynomial and the radial are two of the most commonly used kernel functions. The polynomial kernel is $(\gamma u'v + k)^d$ where γ and k are constants and d is the degree of the kernel function. For $d = 1$ the kernel is called linear kernel. The radial kernel is $\exp(-\gamma|u - v|^2)$. The choice of kernel depends on the data and the invariance of the data, where the different kernels solves different invariances.

2.8 Neural networks, Deep learning

A neural network is designed to replicate the system of neurons that exists in a human brain. A neuron is built up by an axon that sums up the information gathered by a dendritic tree taking in information in different branches. If the dendritic tree receives enough stimulation, it will create a spike to the axon. The axon is then connected to a branch of another dendritic tree and the process repeats itself in a net of neurons.

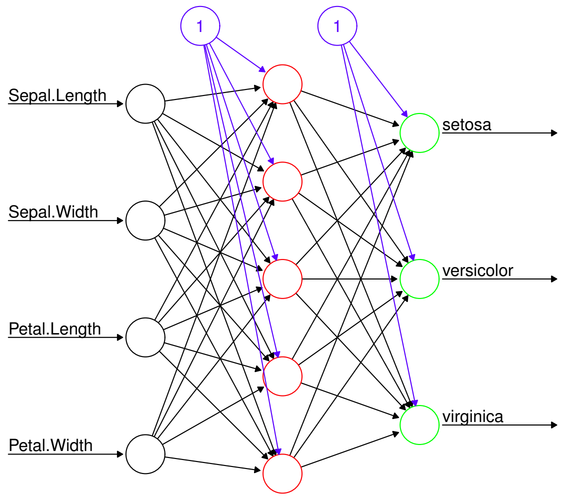


Figure 2.4: Neural network with an input layer (black), a hidden layer (red) and an output layer (green). The blue nodes represents the bias for each layer.

The machine learning method works in the same way. The example neural network in Figure 2.4 is constructed from the iris data using all 4 input features (the black nodes), which are passed through 1 hidden layer of 5 nodes (red). The probability for each class output is finally passed through the output layer (green nodes). The input vector is multiplied by a weight vector, and if this sum exceeds some threshold it will create an output. In each layer l of the neural network, there is a number j of neurons that all take in a vector of i input values, all multiplied by weights. These j neurons create outputs that will be represented as inputs in the next layer of neurons. The amount of neurons in each layer can differ and the structure of the neural net can be more or less advanced. For example, a node in the next layer can skip some of the outputs from the earlier layer and/or sum up weighted outputs from even earlier layers. How this architecture is built depends on the application area, the data, which type of learning is used, among other things.

The weights are described by

$$w_{i,j}^{(l)}, \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

where L is the total number of layers, $d^{(l)}$ is the dimension of layer l and for $i = 0$

the weight is from the bias. In theory there is a threshold connected to a step-function that a neuron needs to exceed in order to create an output, the bias in Figure 2.4 represents this threshold. This step-function is not differentiable in all points, and it is therefore convenient to use some function that is. This function is called an activation function and can be different functions depending on the learning type and application area. When using a supervised learning algorithm for binary classification, the logistic sigmoid function is commonly used. The output from each node is given by

$$z_j^{(l)} = h\left(\sum_{i=0}^{d^{(l-1)}} w_{j,i}^{(l)} z_i^{(l-1)}\right) = h(a_j), \tag{2.12}$$

where $h(\cdot)$ is the activation function. Equation (2.12) is applied recursively, starting on the input $\boldsymbol{z}^{(0)} = \boldsymbol{x} = [x_1 \dots x_{d^{(0)}}]$ ending in the output $\boldsymbol{z}^{(L)} = \boldsymbol{y}$ in layer L. The output is then the prediction of the network.

The machine learning algorithm is a way to minimize the error function according to the different weights. The error function for a binary classification problem is the cross entropy function:

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} [t_n \log(y(\boldsymbol{x}_n, \boldsymbol{w})) + (1 - t_n)\log(1 - y(\boldsymbol{x}_n, \boldsymbol{w}))],$$

where t_n are the target values and $y(\boldsymbol{x}_n, \boldsymbol{w})$ are the outputs. This is the sum of all the N data points, and the function inside the sum will hereby be called $E_n(\boldsymbol{w})$. When using neural networks for regression, the error function can be described as the squared error:

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} \frac{1}{2}(y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n)^2.$$

This is optimized by an algorithm that changes the weights in each iteration until it reaches a minimum. This iteration of weights is described by:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \Delta \boldsymbol{w}^{(\tau)},$$

where τ labels the iteration step. This algorithm often uses some optimization algorithm that uses the gradient of the error function to find a minimum, such as gradient descent, conjugate gradients or quasi-Newton methods. To use these optimization methods, the gradient needs to be calculated. This is done by using a backpropagation method where the chain rule is used to find the derivatives of the different weights. For the output weight the gradient becomes:

$$\frac{\partial E_n}{\partial w_{j,i}^{(L)}} = \frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\partial y(\boldsymbol{x}_n, \boldsymbol{w})}{\partial w_{j,i}^{(L)}} = \frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\partial y(\boldsymbol{x}_n, \boldsymbol{w})}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}^{(L)}} = \frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} h'(a_j) h(a_i), \tag{2.13}$$

where

$$\frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} h'(a_j) = \delta_j.$$

The logistic sigmoid function has the convenience of having the following derivative:

$$\sigma' = \sigma(1 - \sigma),$$

and the derivative of the cross entropy error function is:

$$\frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} = \frac{(y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n)}{(1 - y(\boldsymbol{x}_n, \boldsymbol{w})) y(\boldsymbol{x}_n, \boldsymbol{w})}.$$

With these results, the weights in the hidden layers can be decided. The last hidden layer gets the following gradient:

$$\begin{aligned}
\frac{\partial E_n}{\partial w_{j,i}^{(L-1)}} &= \frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\partial y(\boldsymbol{x}_n, \boldsymbol{w})}{\partial w_{j,i}^{(L-1)}} \\
&= \frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\partial y(\boldsymbol{x}_n, \boldsymbol{w})}{\partial a_k} \frac{\partial a_k}{\partial w_{j,i}^{(L-1)}} \\
&= \frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\partial y(\boldsymbol{x}_n, \boldsymbol{w})}{\partial a_k} \frac{\partial a_k}{\partial h(a_j)} \frac{\partial h(a_j)}{\partial w_{j,i}^{(L-1)}} \\
&= \frac{\partial E_n}{\partial y(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\partial y(\boldsymbol{x}_n, \boldsymbol{w})}{\partial a_k} \frac{\partial a_k}{\partial h(a_j)} \frac{\partial h(a_j)}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}^{(L-1)}},
\end{aligned}$$

where the first two partial derivatives are exactly the same as in (2.13) and the last equation is the same but for this layer. By using all the outputs with their respective weights, the equation can then be written:

$$\frac{\partial E_n}{\partial w_{j,i}^{(L-1)}} = h(a_i) h'(a_j) \sum_k \delta_k w_{k,j}^{(L-1)},$$

and by using the following simplification:

$$\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}^{(L-1)},$$

the following formula can be used for all the hidden layers and the output layer:

$$\frac{\partial E_n}{\partial w_{j,i}^{(l)}} = \delta_j z_i,$$

where the calculations use the results from the recently calculated layers until resulting in the gradient according to the weights applied to the input vector. When using the neural network algorithm, the weights are initially chosen at random. This is because the error function is not a completely convex function according to the weights and therefore multiple local minima can be found. There is no way to tell if the local minimum that was actually found is a global minimum therefore the algorithm needs to run multiple times to find different local minima. The best local minimum is hopefully the global minimum or close to it. In order to not achieve the same minimum every run, the initial weights are chosen at random.

The neural net can be designed with multiple hidden layers to become able to handle non-linear problems. This is called a deep neural net and is the most common kind of Deep Learning. These kinds of neural networks can model complex data with fewer points than a more shallow neural network can. Deep neural networks can be designed in many different ways depending on the target application. The research on this is extensive and since deep learning is a very popular area, this research is updated continuously. The neural networks can be created in different styles with different weights between the different layer nodes etc. (Murphy, 2012, pp. 995-1007)

2.9 ROC and AUC

Receiving operating characteristics, ROC, is a way of visualizing classification performance. The method is typically applied to binary classification problems, but there are ways of extending it to multiple classes.

A classifier's performance is typically measured by how many instances it predicts correctly. If the data is skewed, which it often is in reality, this can be very problematic. An imbalanced data set can produce a very high accuracy by just classifying every instance as the majority class if the minority class is very small. In ROC the false positive vs. false negative tradeoff is used instead (Murphy, 2012, p. 180). That is, given a classifier and an instance, there are four possible outcomes: if the instance is positive and classified as positive it is considered to be a *true positive*, if the instance is instead classified as negative it is considered a *false negative*, if the instance is negative and classified as negative it is considered a *true negative*, while if it is classified as positive it is considered a *false positive*. These results are typically presented in a confusion matrix, see Table 2.1.

The ROC space consists of the *true positive rate*, *tpr*, on the y-axis and the *false positive rate*, fpr, on the x-axis, and a discrete classifier will be represented by a point. Hence the point (0,1) in ROC space represents perfect classification (100% true positive rate and 0% false positive rate) while the points (0,0) and (1,1) represent the extremes where all instances are classified as either negative or positive respectively. The diagonal line $x = y$ in the ROC space represents randomly guessing the class. Thus a classifier should produce a point in the upper triangle as a point in the lower triangle is to be considered worse than randomly guessing. By varying the threshold, i.e. the probability that needs to be exceeded by the classifier to classify an instance as positive, different points in the ROC space will be produced, which can be traced to a curve. The best threshold is commonly given by Youden's index, that is the threshold that maximizes the equation $tpr + 1 - fpr$ (Youden, 1950). This is the threshold that classify a lot of true positives but few false positive and is commonly recommended as the probability threshold when predicting new data points.

ROC curves are insensitive to changes in the class distribution which make them very useful when comparing performance for classifiers on skewed data sets. To compare classifiers, the quality of the ROC curve is summarized as a single scalar value representing the expected performance by calculating the Area Under the Curve, AUC. As the ROC operates in the unit square, the AUC will always have a value between 0 and 1. A higher AUC score is better, and a score less than 0.5 is worse than guessing. (Fawcett, 2006)

		Prediction outcome		
		p	**n**	**total**
actual value	$\mathbf{p'}$	True Positive	False Negative	P'
	$\mathbf{n'}$	False Positive	True Negative	N'
	total	P	N	

Table 2.1: Confusion matrix

Chapter 3

Insurance

3.1 Data

The data used in this and the following two chapters were provided by van der Putten and van Someren. (2000) at the Dutch data mining company Sentient Machine Research for the CoIL 2000 Challenge and was obtained through the web page of Lichman (2013). The data contains information about customers consisting of 86 features. Features 1-43 contain socio-demographic data and features 44-86 contain customer specific product ownership, where feature 86 is the target feature. The socio-demographic data is derived from zip area codes and is an average of the population living in that specific area, meaning that all customers living in the same area will have the same socio-demographic features. However, the zip area codes are not included in the data set. This is important to note to be able to correctly interpret the data since it only provides a probable image of the customer and not the real values for each specific customer. The product ownership features, column 44-86, contains information about which insurances the customer currently own and their costs. The target feature, feature 86 "CARAVAN: Number of mobile home policies", has only two classes, class 0 or class 1, i.e. the customer either has or does not have a caravan insurance. Since the problem is to predict the presence of a caravan policy, the desired class is 1. Information of all the attributes can be found in Appendix A.

The data was already split in two parts, a training data set and an evaluation data set. The training data consists of 5822 data points with the target feature included. In the training data there are 348 instances of class 1, which is roughly one sixteenth of the data, making the data set quite imbalanced. The training data is split into a training set consisting of 75% (4366 observation) of the original data, and a validation set consisting of the remaining 25% (1456 data points). The models are trained on the training set and their prediction performance on the validation set are measured using ROC and AUC. The training data is split randomly into the two subsets and the same subsets are used for training and validating every model.

The evaluation set consist of 4000 data points where the target feature is left out. The target features associated with the validation set are found in a separate data set. In this data set there are 238 instances of class 1. Comparing the prediction to the true target results provides a hit rate on how well the models perform.

3.2 Modelling

All implementation has been done in R (R Core Team, 2016). The *train*-function, found in the *caret*-package (from Jed Wing et al., 2016), has been used for training all the models except the naïve Bayes. The training has been done using 10-folded cross validation repeated 5 times. The models are tuned over different parameters, see below, and the model with the best parameters for each method is chosen by using AUC as performance metric.

3.2.1 Training the models and tuning the parameters

This section explains how the different methods were trained and over which parameters they were tuned.

Training and tuning the generalized linear model

The generalized linear model is optimized by maximum likelihood using Fisher's scoring to find the optimal coefficients for all the features. There are no parameters chosen by the user and no tuning is required.

Training and tuning naïve Bayes

The naïve Bayes is trained by calculating the class probabilities. There are no parameters chosen by the user and no tuning is required.

Training and tuning k-NN

To find the optimal k, the *train*-function is set to tune over $k = 1, 3, 5, ..., 69$.

Training and tuning decision tree

The decision tree is trained by finding the optimal splits using the Gini-index. No parameters are chosen by the user, and no tuning is required.

Training and tuning the random forest

The random forest is tuned over how many features are randomly chosen for the splits in each tree m. The chosen interval is $m = 1, ..., 50$, therefore including the recommended value $m = \sqrt{p} \approx 9$ from Section 2.6.2.

Training and tuning support vector machines

Three different SVMs are trained using three different kernel functions: linear, radial basis and polynomial kernel functions. The parameter intervals are arbitrarily chosen with consideration to typical values as well as keeping a manageable computational time.

Using the linear kernel function, the only tunable parameter is the cost function C, which is set to be $C = 10^n$ for $n = -20, ..., 5$. The interval chosen for the linear kernel is wider than for the other kernels due to its shorter computational time.

The radial basis kernel is tuned over $\sigma = \dim(trainingset) \cdot 2^m$ for $m = -5, ..., 5$ and the cost parameter $C = 10^n$ for $n = -5, ..., 5$.

The polynomial kernel is tuned over the degree parameter $d = 2, ..., 8$ and cost parameter $C = 10^n$ for $n = -5, ..., 5$. The scaling parameter $a = 1$ is held constant in all versions of the model.

Training and tuning the neural net

The neural nets are trained with 1 hidden layer, with a range from 10 to 120 nodes and a stepping size of 10.

3.2.2 Validation

The best models are being validated on the validation data set by generating the ROC-curves to compare the AUC and selecting the recommended threshold, as shown in the ROC curves. The curves are generated by the *pROC*-package (Robin et al., 2011). The models are then retrained using all the 5822 data training points and the final models are applied to the untouched evaluation data set consisting of 4000 data points. Comparing the predicted result with the true result generates a hit rate for each of the models.

The models for the plots are trained on the training data set and used for predicting the classes of the validation data set. When computing the hit rate the models have been retrained with the best tuned parameters and set to predict the classes of the evaluation data set, given the selected threshold for each model.

3.3 Results

The results in this section are generated using the insurance data without any alterations.

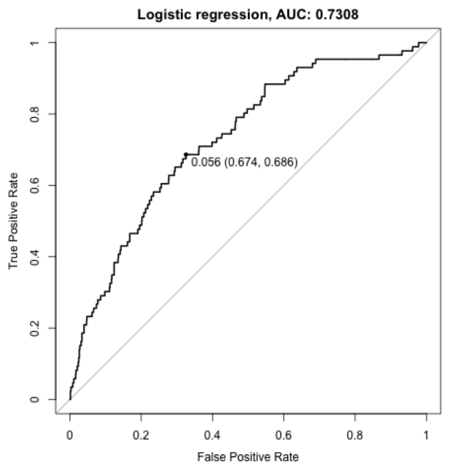


Figure 3.1: **Generalized linear model:**
ROC-curve for the generalized linear model on unaltered data.

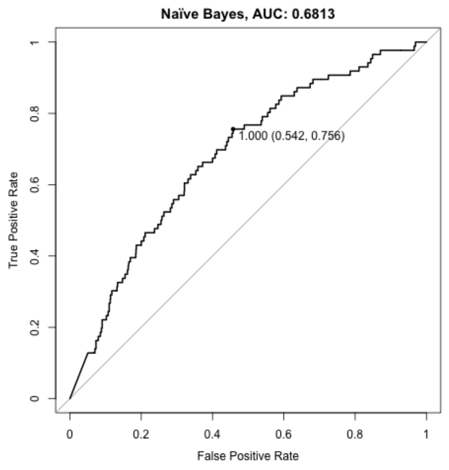


Figure 3.2: **Naïve Bayes:**
ROC-curve for Naïve Bayes on unaltered data.

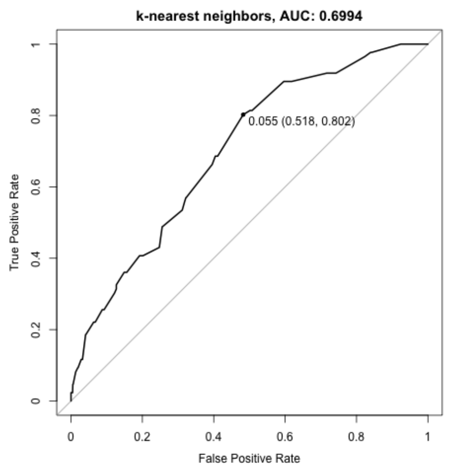


Figure 3.3: **k-Nearest Neighbors:** ROC-curve for k-nearest neighbors on unaltered data.

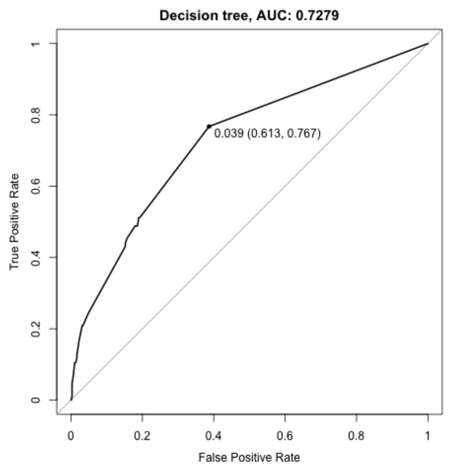


Figure 3.4: **Decision tree:** ROC-curve for the decision tree on unaltered data.

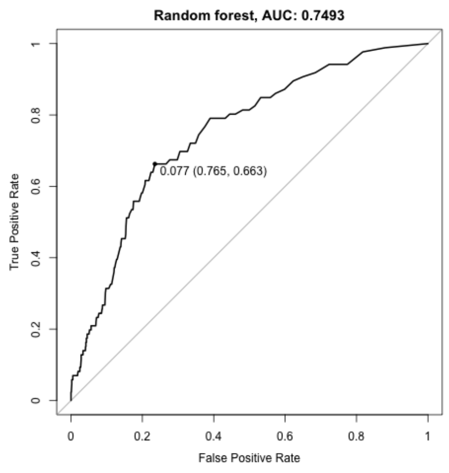


Figure 3.5: **Random forest:** ROC-curve for the random forest on unaltered data.

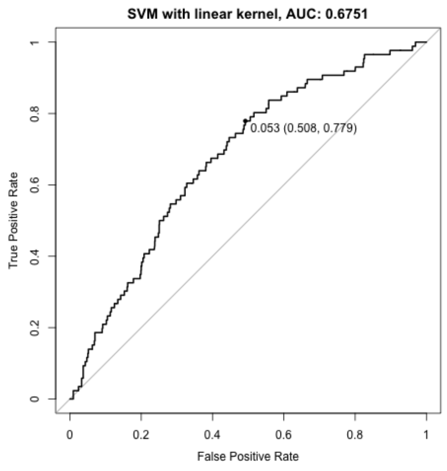


Figure 3.6: **SVM with linear kernel:** ROC-curve for the support vector machine with linear kernel function on unaltered data.

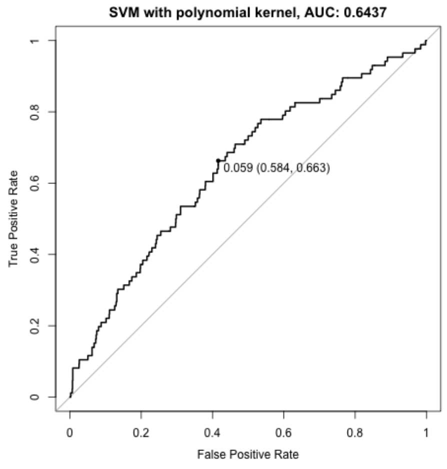


Figure 3.7: **SVM with polynomial kernel:** ROC-curve for the support vector machine with polynomial kernel function on unaltered data.

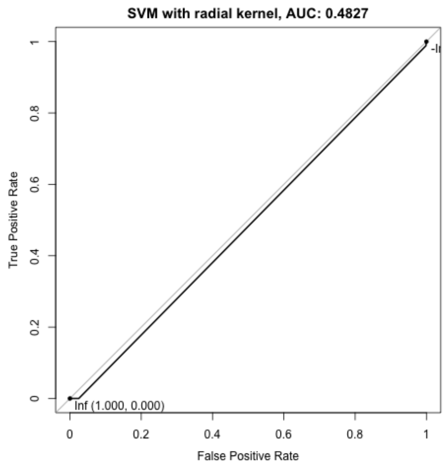


Figure 3.8: **SVM with radial kernel:** ROC-curve for the support vector machine with radial kernel function on unaltered data.

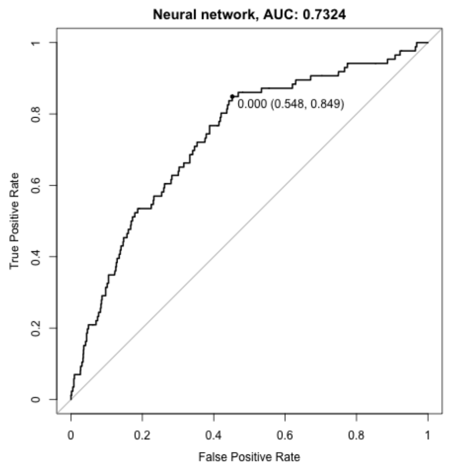


Figure 3.9: **Neural network:** ROC-curve for the neural network on unaltered data.

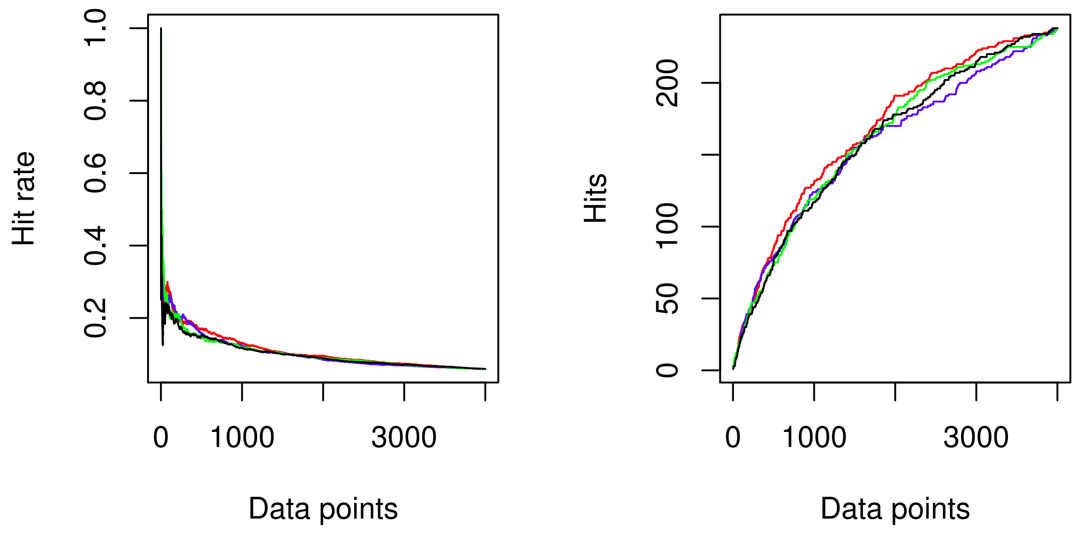


Figure 3.10: Plot of hit rate and amount of hits when increasing the amount of data points chosen. The red line is Generalized linear model, the green line is Random forest, the blue line is Decision tree and the black line is Neural network.

	AUC	Hit rate
GLM	0.7308	149/1353=0.110
Naïve Bayes	0.6813	151/1626=0.093
k-nearest neighbors	0.6994	176/1908=0.092
Decision tree	0.7279	160/1617=0.099
Random forest	0.7493	120/1012=0.119
SVM - linear kernel	0.6751	238/4000=0.060
SVM - polynomial kernel	0.6437	236/3994=0.059
SVM - radial kernel	0.4827	238/4000=0.060
Neural network	0.7324	168/1761=0.096

Table 3.1: AUC and hit rate results for the different methods.

3.4 Analysis

Looking at the AUC for the different methods, we see that they all show similar performance with an AUC around 0.7, see the left column in Table 3.1. The exception is the support vector machine with radial kernel function that seem to predict worse than random guessing. The other two support vector machines perform better but are still performing worse than the other methods. This suggests that the chosen kernel functions are not ideal to use on the data set at hand. The machine learning method that shows the best suggested performance is the random forest with an AUC of 0.7493 followed by the neural network (0.7324) and the decision tree (0.7279). The generalized linear model performs surprisingly well with an AUC of 0.7308.

Comparing the hit rates in Table 3.1, the random forest performs best with a hit rate of 11.9%, closely followed by the generalized linear model with a hit rate of 11%. The neural network and the decision tree do not seem to perform as well on the new data with a hit rate just under 10%. The support vector machines perform poorly on the new data with a hit rate of 6.0%, 5.9% and 6.0% respectively, which was to be expected given their AUC.

Chapter 4

Rebalancing the training data

4.1 Theory

SMOTE stands for Synthetic Minority Over-sampling Technique. SMOTE is an over sampling and under sampling method created by Chawla et al. (2011). Over sampling and under sampling is used when the data set is not symmetric and the target class is of minority. A data set is symmetric for when there are equally amount of data points corresponding to both classes in a binary classification problem. When the data set is not symmetric there is a majority class and a minority class. The method combines under sampling of the majority class with an over sampling technique, using a k-NN method, for the minority class. The under sampling part will randomly select data points of the majority class and remove them from the data set. The over sampling part uses the k-nearest minority class neighbors of a minority class data point to create new data points on the lines joining the neighbors to the original point. The new point is placed at a random distance on the line. If oversampling by 200% is used then the two nearest neighbors are used to create new points for all minority class data points, which will result in 300% as many minority class data points as before. (Chawla et al., 2011)

4.2 Modelling

To examine the effects of the imbalance of the target feature we rebalance the data with the SMOTE. Using the *SMOTE*-function provided by the *DMwR*-package (Torgo, 2010), three new data sets are generated from the original data set. In the new data sets the desired class, class 1, make up 30%, 50% and 70%, respectively. These data sets are referred to as 30% SMOTE, 50% SMOTE and 70% SMOTE. The generated data sets all contain 2620 data points. All models were applied to the new data sets and tuned over the same parameters as in Section 3.2.1.

4.3 Results

The results in this section are generated using the three data sets generated by the SMOTE in Section 4.2. When training the final models for hit rate, the whole data set is taken into account and rebalanced using the SMOTE. The ROC-curves for every method is presented below, and a complete list of their respective AUC-values is presented in Table 4.1.

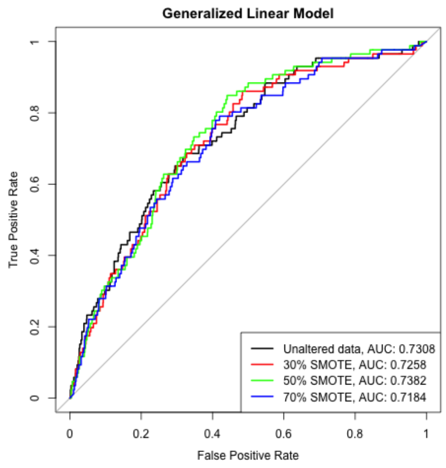


Figure 4.1: **Generalized linear model:** ROC-curves and corresponding AUC for the generalized linear model on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

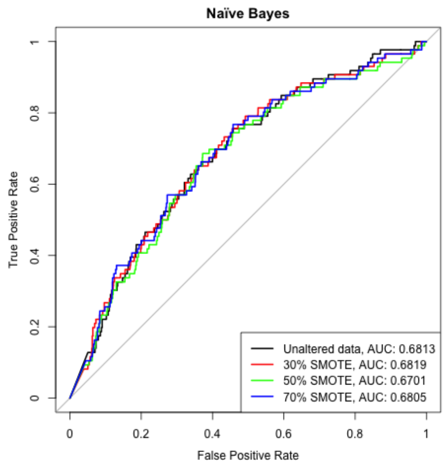


Figure 4.2: **Naïve Bayes:** ROC-curves and corresponding AUC for naïve Bayes on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

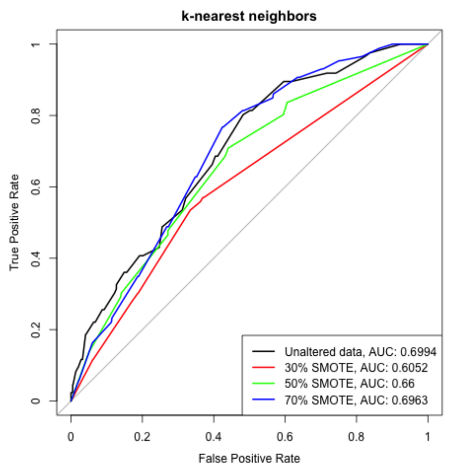


Figure 4.3: **k-Nearest Neighbors:** ROC-curves and corresponding AUC for k-nearest neighbors on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

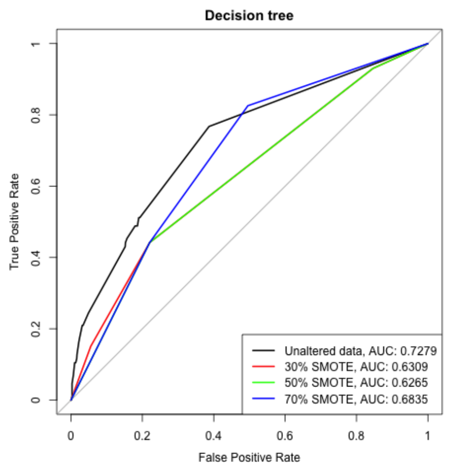


Figure 4.4: **Decision tree:** ROC-curves and corresponding AUC for the decision tree on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

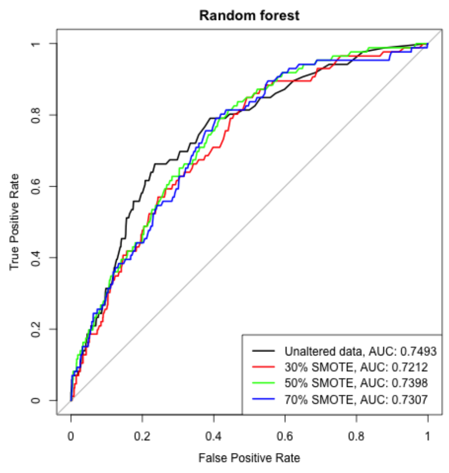


Figure 4.5: **Random forest:** ROC-curves and corresponding AUC for the random forest on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

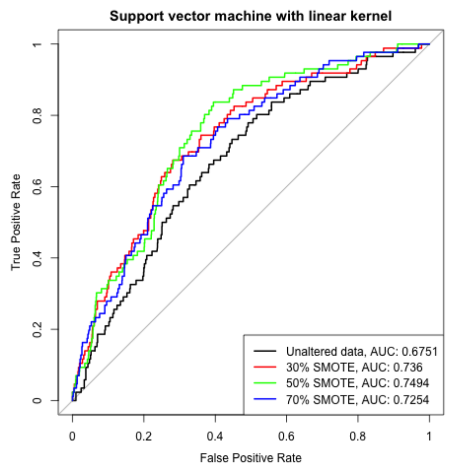


Figure 4.6: **SVM with linear kernel:** ROC-curves and corresponding AUC for the support vector machine with linear kernel function on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

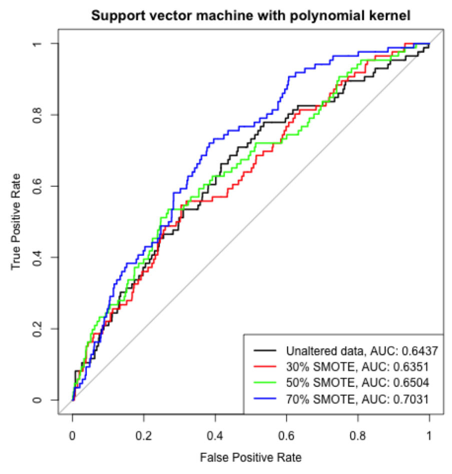


Figure 4.7: **SVM with polynomial kernel:** ROC-curves and corresponding AUC for the support vector machine with polynomial kernel function on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

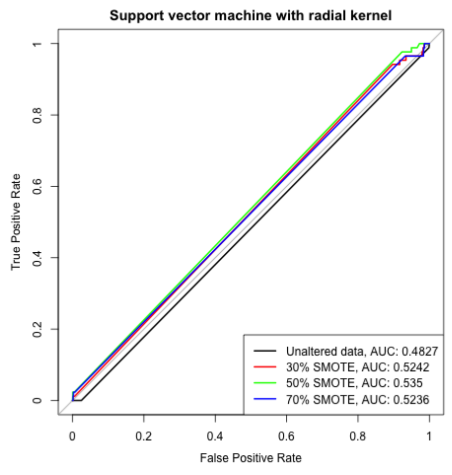


Figure 4.8: **SVM with radial kernel:** ROC-curves and corresponding AUC for the support vector machine with radial kernel function on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

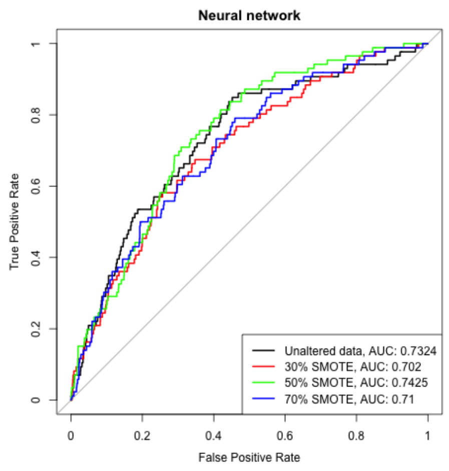


Figure 4.9: **Neural network:** ROC-curves and corresponding AUC for the neural network with on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

	AUC			
	Normal data	SMOTE		
		30%	50%	70%
GLM	0.7308	0.7258	**0.7382**	0.7184
Naïve Bayes	0.6813	**0.6819**	0.6701	0.6805
k-nearest neighbors	**0.6994**	0.6052	0.6600	0.6936
Decision tree	**0.7279**	0.6309	0.6265	0.6835
Random forest	**0.7493**	0.7246	0.7398	0.7307
SVM - linear kernel	0.6751	0.7360	**0.7494**	0.7254
SVM - polynomial kernel	0.6437	0.6351	0.6504	**0.7031**
SVM - radial kernel	0.4827	0.5242	**0.5350**	0.5236
Neural network	0.7324	0.7020	**0.7425**	0.7100

Table 4.1: AUC result for the different methods with the different data sets.

	Hits/Points over threshold (PoT) = Hit rate			
	Normal data	SMOTE		
		30%	50%	70%
GLM	**149/1353 = 0.110**	178/2045 = 0.087	164/1809 = 0.091	161/1795 = 0.090
Naïve Bayes	**151/1626 = 0.093**	162/1891 = 0.086	135/1520 = 0.089	157/1910 = 0.082
k-nearest neighbors	176/1908 = 0.092	125/1423 = 0.088	160/1893 = 0.085	**30/285 = 0.105**
Decision tree	160/1617 = 0.099	**109/937 = 0.116**	**109/937 = 0.116**	172/2020 = 0.085
Random forest	120/1012 = 0.119	97/717 = 0.135	64/406 = 0.158	**46/224 = 0.205**
SVM - linear kernel	238/4000 = 0.060	147/1392 = 0.106	168/1915 = 0.088	**45/277 = 0.162**
SVM - polynomial kernel	236/3994 = 0.059	117/1211 = 0.097	238/4000 = 0.060	**101/862 = 0.117**
SVM - radial kernel	238/4000 = 0.060	224/3716 = 0.060	227/3772 = 0.060	**9/45 = 0.2**
Neural network	168/1761 = 0.096	129/1399 = 0.092	**130/1265 = 0.102**	125/1284 = 0.097

Table 4.2: Hit rates for the best thresholds for the different methods with the different data sets.

4.4 Analysis

When using SMOTE on the data, we see changes in the results. Most noticeable is the performance of the SVM with linear kernel. When there is an equal amount of zeros and ones in the target class, the AUC is 0.7494, which is as good as for the random forest for the unaltered data. The 30% and 70% SMOTE data show similar improvement in performance for the SVM with linear kernel. The SVM with polynomial kernel shows an increased performance on the 70% SMOTE data set but not for the 30% and the 50% SMOTE data. For the simpler methods, generalized linear model, naïve Bayes and k-NN, the rebalanced data shows no real improvement from the unaltered data set. The same goes for the random forest. However the decision tree seems to perform worse on the rebalanced data than the original.

On most of the models, the hit rates for the best threshold of the models score higher on the rebalanced data sets. For a majority of the models, the highest scored hit rate is achieved when trained on the 70% SMOTE data set. The improved hit rates are most apparent for the support vector machines whose hit rates for the 70% SMOTE are several percentage points above the other three data sets. For this data set the SVM with radial kernel score the second highest hit rate of 20%, only outperformed by the random forest on the same data set which scores a hit rate of 20.5%. Most noticeable is that the random forest score the highest hit rate for every data set, regardless of how imbalanced the data might be. With the exception of the support vector machines, the different methods seem to handle the skewness in the original data rather well. The generalized linear model and neural network also gets higher AUC for 50% SMOTE. The other models perform the same or worse.

Chapter 5

Feature reduction

5.1 Modelling

When creating these models, the amount of data can be a problem as well. Since this data set only contains 5822 data points, one can say that this is too few to make predictions based on 85 features. Some of the input features may also be unnecessary or contribute to adding noise to the predictions. To examine these potential effects we reduce the number of input features used for making the predictions. From the generalized linear model (GLM) we can retrieve the input parameters that has the highest certainty, i.e. the lowest p-value. In this way 15 input features were selected for each data set (the original set and the three rebalanced sets), and all models were trained on the selected subset of input features. The 15 features chosen for the 4 data sets can be found in Appendix B. The models are tuned over the same parameters as in Section 3.2.1 with the exception of the random forest, where the parameter is set over the interval $m = 1, ..., 15$ since the parameter has to be less than or equal to the number of input parameters.

5.2 Results

The results in this section are generated using the original data with reduced input features as well as the SMOTE data with reduced input features. The ROC-curves for every method is presented below, and a complete list of their respective AUC-values is presented in Table 5.1.

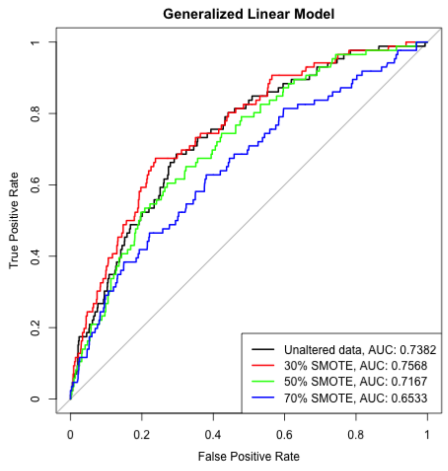


Figure 5.1: **Generalized linear model:** ROC-curves and corresponding AUC for the generalized linear model on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

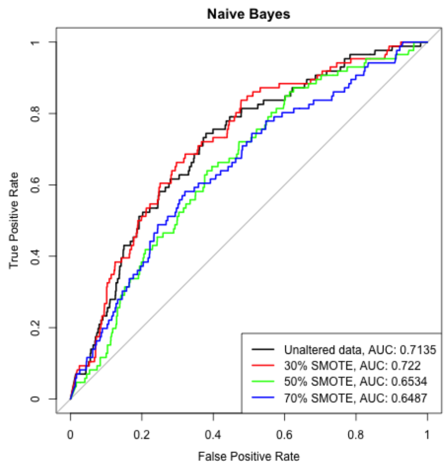


Figure 5.2: **Naïve Bayes:** ROC-curves and corresponding AUC for naïve Bayes on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

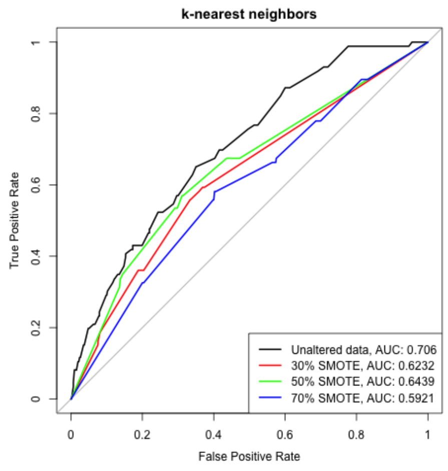


Figure 5.3: **k-Nearest Neighbors:** ROC-curves and corresponding AUC for k-nearest neighbors on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

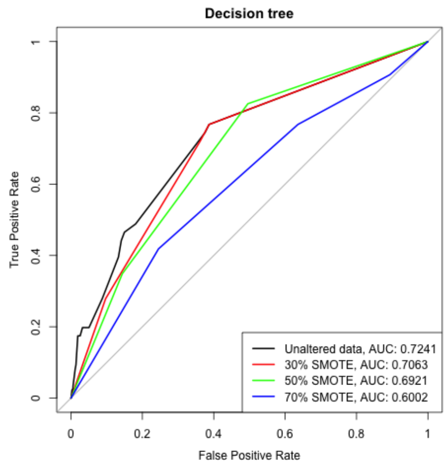


Figure 5.4: **Decision tree:** ROC-curves and corresponding AUC for the decision tree on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

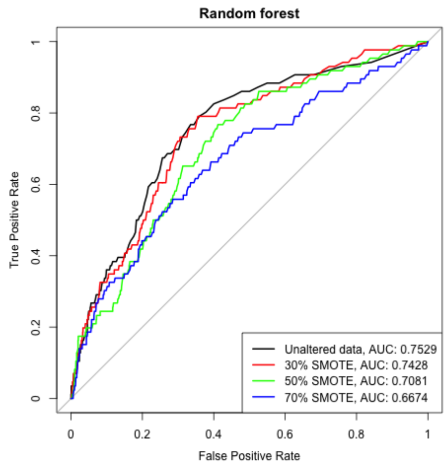


Figure 5.5: **Random forest:** ROC-curves and corresponding AUC for the random forest on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

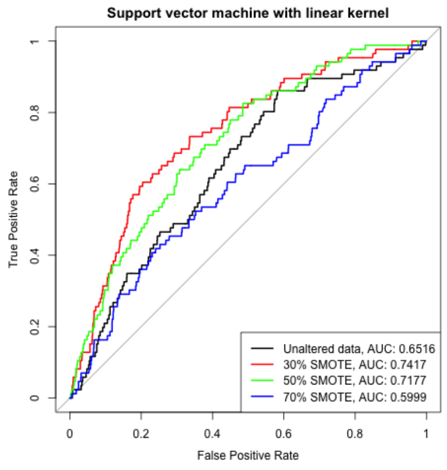


Figure 5.6: **SVM with linear kernel:** ROC-curves and corresponding AUC for the support vector machine with linear kernel function on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

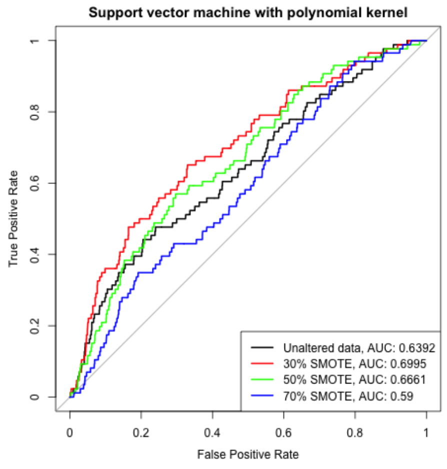


Figure 5.7: **SVM with polynomial kernel:** ROC-curves and corresponding AUC for the support vector machine with polyomial kernel function on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

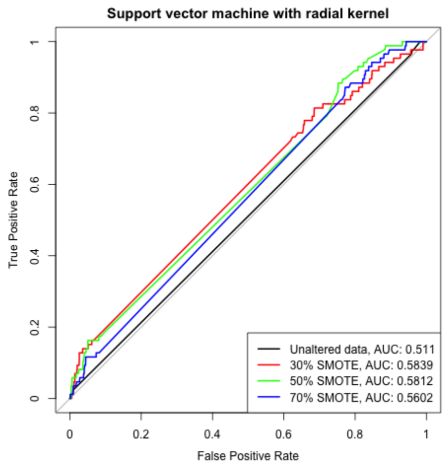


Figure 5.8: **SVM with radial kernel:** ROC-curves and corresponding AUC for the support vector machine with radial kernel function on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

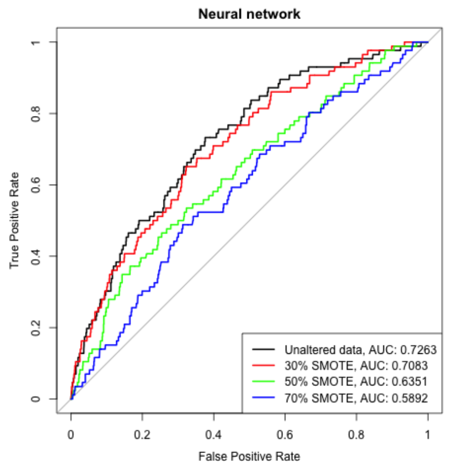


Figure 5.9: **Neural network:** ROC-curves and corresponding AUC for the neural network on unaltered data (black), 30% SMOTE data (red), 50% SMOTE data (green) and 70% SMOTE data (blue).

	AUC			
	Normal data	SMOTE		
		30%	50%	70%
GLM	0.7382	**0.7568**	0.7167	0.6533
Naïve Bayes	0.7135	**0.7220**	0.6534	0.6487
k-nearest neighbors	**0.7060**	0.6232	0.6439	0.5921
Decision tree	**0.7241**	0.7063	0.6921	0.6002
Random forest	**0.7529**	0.7428	0.7081	0.6674
SVM - linear kernel	0.6516	**0.7417**	0.7177	0.5999
SVM - polynomial kernel	0.6392	**0.6995**	0.6661	0.5900
SVM - radial kernel	0.5110	**0.5839**	0.5812	0.5602
Neural network	**0.7263**	0.7083	0.6351	0.5892

Table 5.1: AUC result for the different methods with the different data sets with reduced number of features.

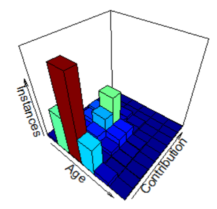
	Hits/PoT=Hit rate			
	Normal	SMOTE		
	data	30%	50%	70%
the generalized linear model	154/1327 = 0.116	**127/1080 = 0.117**	143/1307 = 0.109	158/1616 = 0.098
Naïve Bayes	**176/1805 = 0.098**	148/1653 = 0.090	182/2197 = 0.083	142/1685 = 0.084
k-nearest neighbors	**158/1486 = 0.106**	129/1385 = 0.093	110/1246 = 0.088	136/1643 = 0.083
Decision tree	160/1617 = 0.099	160/1617 = 0.099	158/1533 = 0.103	**104/916 = 0.114**
Random forest	**152/1392 = 0.109**	150/1385 = 0.108	161/1737 = 0.093	115/1208 = 0.095
SVM - linear kernel	238/4000 = 0.060	205/2846 = 0.072	172/2020 = 0.085	**65/618 = 0.105**
SVM - polynomial kernel	**4/11 = 0.364**	126/1382 = 0.091	116/1160 = 0.1	40/384 = 0.104
SVM - radial kernel	220/3790 = 0.060	**23/149 = 0.154**	23/194 = 0.085	31/258 = 0.120
Neural network	162/1764 = 0.092	**119/1249 = 0.095**	128/1494 = 0.086	107/1198 = 0.089

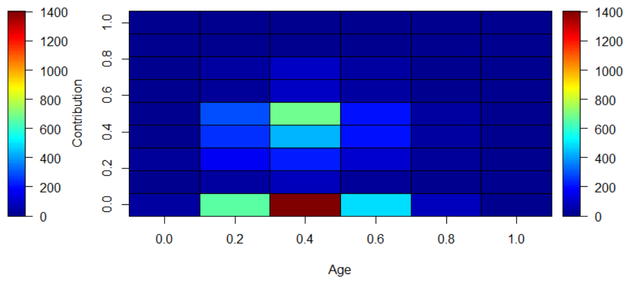
Table 5.2: Hit rates for the best thresholds for the different methods with the different data sets with reduced number of features.

5.3 Analysis

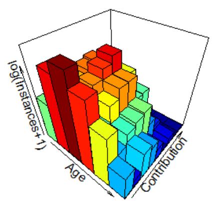
Reducing the number of input features from 85 to 15 did not seem to improve the prediction performance noticeably for any of the models. Looking at the AUC values in Table 5.1, the models get similar performance as when training on all the available input features. Combining the reduced number of features and the SMOTE, the models in most cases get a lower AUC value than before. Looking at the hit rates in Table 5.2, the results coincide with the AUC values. For the normal data set the simpler models show a slight improvement in both AUC values and hit rate scores but perform worse on the SMOTE data sets. Even though there was not a significant improvement in performance, there was an improvement in computational time for the more time consuming methods, especially the neural net.

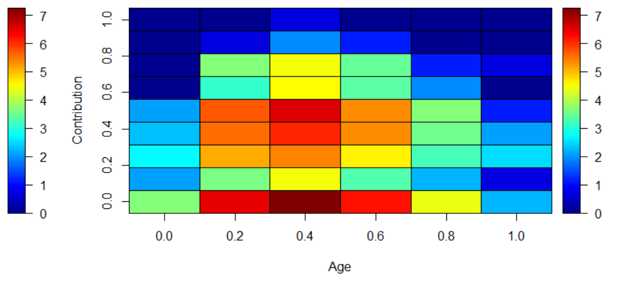
The p-value is different for the different data sets. There are only two features that are in the top 15 in all four data sets, and these are number 4, average age of people living in the customers zip code area, and number 59, contributions to fire policies. Kendall's tau of the two features are 0.519 and the Spearman's rho is 0.581. Figure 5.10 show the histogram for the two variables plotted against each other.





(a) (b)





(c) (d)

Figure 5.10: This figure contains plots of feature 4: average age of people living in the customers zip code area, and feature 59: customers contributions to fire policies. (a) represents the 3D-histogram for instances over feature 4 and 59. (b) represents the same as (a) viewed in the direction of the z-axis. (c) represents the 3D-histogram for the logarithm of instances plus one over feature 4 and 59. (d) represents the same as (c) viewed in the direction of the z-axis.

Chapter 6

Credit

6.1 Data

The data used in this chapter was provided by Yeh and Lien (2009) for data mining research on predicting probability of credit defaults and contains information about customers and their credit history. It was obtained through the web page of Lichman (2013). The data consists of 30 000 data points with 24 features, including the binary response variable (Yes = 1, No = 0) default payment.

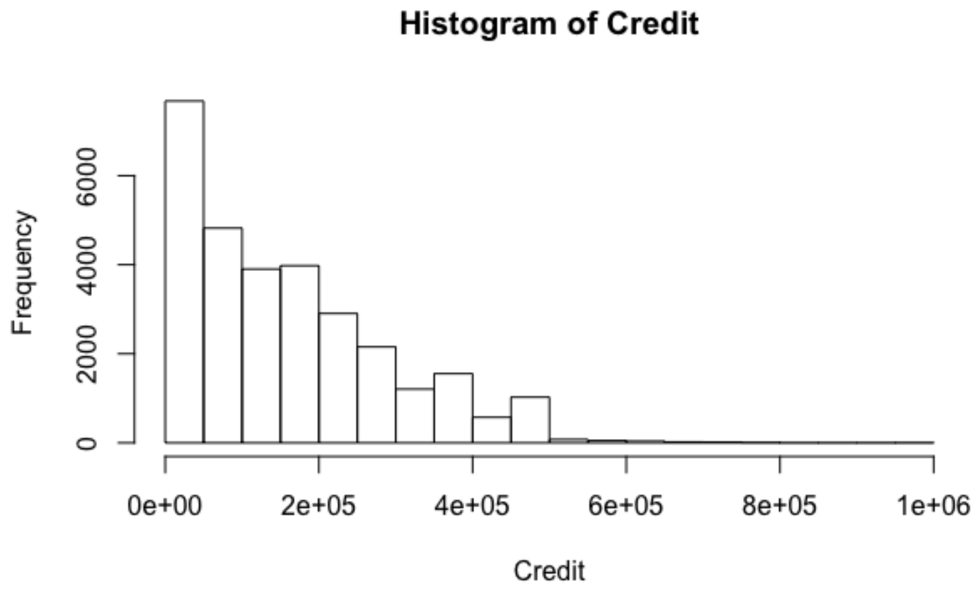


Figure 6.1: Range and frequency of the credit limit of the customers.

The range of credit lies between 10 000 and 1 000 000 NT dollars and is presented in Figure 6.1. A specification for all the features are found in Appendix C.

The data is split into three parts: a training set consisting of 60% (18 000 data points) of the original data, a validation set consisting of 20% (6000 data points) of the original data, and an evaluation set consisting of the remaining 20% (6000 data points) of the original data. The evaluation data set is considered unknown and not used for any training. Instead it is used to give an idea of how well the trained models perform by producing a hit rate.

There is some imbalance in the credit data set as the target class makes up less than a quarter of the data points. The 50% SMOTE approach is applied to the training data set, resulting in 24 228 data points where 50% are of the target class.

6.2 Modelling

The modelling is done in the same way as in Chapter 3. The SMOTE 50% is also used to create a small insight in how the machine learning methods work for another data set. Since there are only 25 features and 30 000 data points there will not be any feature reduction. Due to the low performance compared to computational time of the support vector machines in earlier chapters, these methods are excluded from this chapter for time saving purposes. The methods are tuned and trained in the same way as in Section 3.2.1 over the following grids: for k-NN $k = 1, 3, 5, \ldots, 201$; for random forest $m = 1, 2, 3, \ldots, 20$; for neural network there is 1 hidden layer with a range of 5 to 250 nodes and a stepping size of 5.

6.3 Results

The results in this section are generated using unaltered data set and the 50% SMOTE data set. When training the final models for the hit rate, the whole data set is taken into account and rebalanced using the SMOTE. The ROC-curves for every method is presented below and a complete list of their respective AUC-values is presented in Table 6.1.

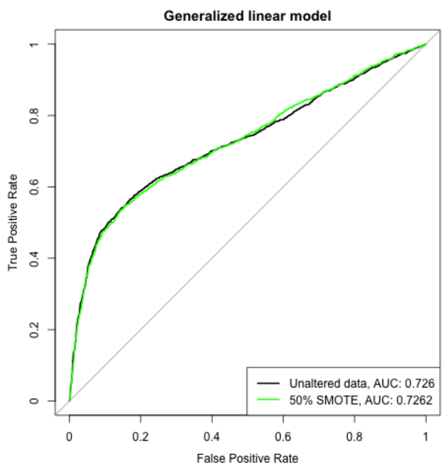


Figure 6.2: **Generalized linear model:**
ROC-curves and corresponding AUC for the generalized linear model on unaltered data (black) and 50% SMOTE data (green).

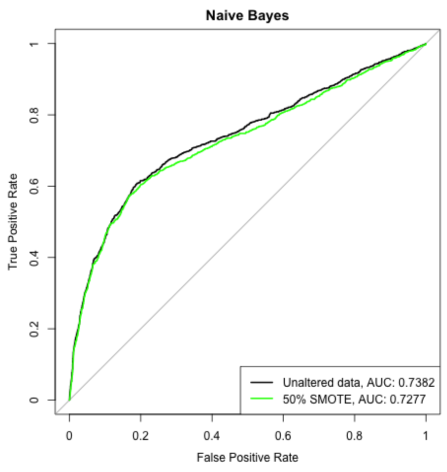


Figure 6.3: **Naïve Bayes:**
ROC-curves and corresponding AUC for naïve Bayes on unaltered data (black) and 50% SMOTE data (green).

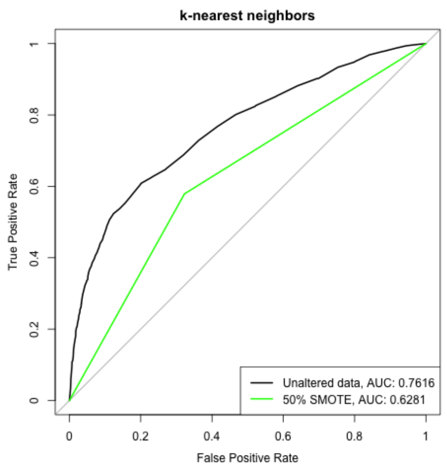


Figure 6.4: **k-Nearest Neighbors:**
ROC-curves and corresponding AUC for k-nearest neighbors on unaltered data (black) and 50% SMOTE data (green).

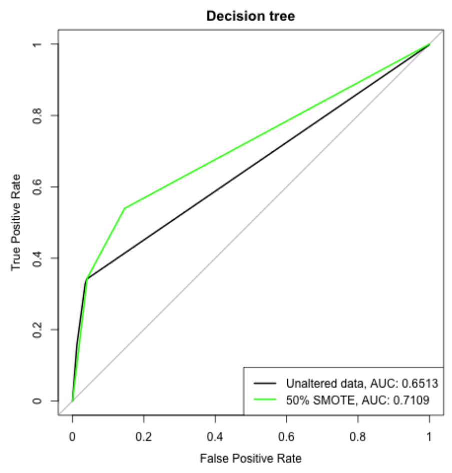


Figure 6.5: **Decision tree:**
ROC-curves and corresponding AUC for the decison tree on unaltered data (black) and 50% SMOTE data (green).

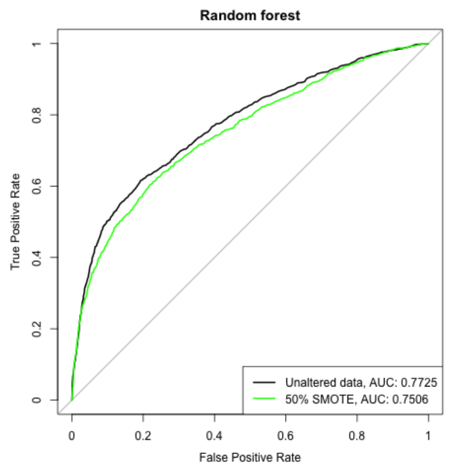


Figure 6.6: **Random forest:**
ROC-curves and corresponding AUC for the random forest on unaltered data (black) and 50% SMOTE data (green).

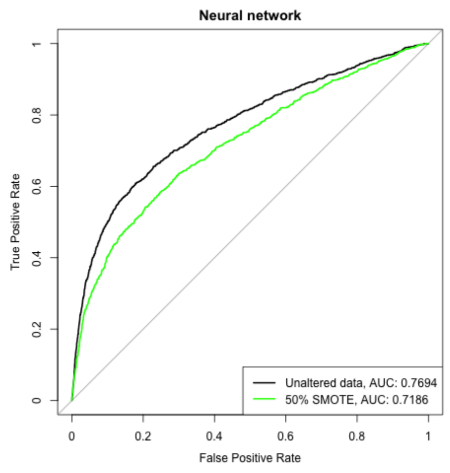


Figure 6.7: **Neural network:**
ROC-curves and corresponding AUC for the neural network on unaltered data (black) and 50% SMOTE data (green).

	AUC	
	Unaltered data	50% SMOTE
GLM	0.7260	**0.7262**
Naïve Bayes	**0.7382**	0.7277
k-nearest neighbors	**0.7616**	0.6281
Decision tree	0.6513	**0.7109**
Random forest	**0.7725**	0.7506
Neural network	**0.7694**	0.7186

Table 6.1: AUC result for the different methods with the different data sets.

	Hits/PoT=Hit rate	
	Unaltered data	50% SMOTE
GLM	698/1421 = 0.491	**193/266 = 0.726**
Naïve Bayes	738/1547 = 0.477	**509/893 = 0.570**
k-nearest neighbors	**750/1700 = 0.441**	494/1318 = 0.375
Decision tree	615/1082 = 0.568	**486/743 = 0.654**
Random forest	788/1737 = 0.454	**542/897 = 0.604**
Neural network	801/1782 = 0.449	**543/907 = 0.599**

Table 6.2: Hit rates for the best thresholds for the different methods with the different data sets.

6.4 Analysis

It can be seen in Table 6.1 that the the methods trained on the unaltered data are often the best versions. It is only for the GLM and the decision tree that the SMOTE version outperforms the unaltered version according to their AUC. Most noticeable is the difference between the two versions of k-NN. The table also shows that the best models are the random forest and after that the neural network, both with the unaltered data. However, in Table 6.2 we see that it is the opposite for hit rates at the best threshold. The models that train on SMOTE data show the best hit rates. The winning model is the GLM followed by the decision tree, both with the rebalanced data.

Chapter 7

Discussion

7.1 Summary analysis

The method that shows the best overall performance is the random forest that score among the top AUCs for every data set. Trying the method on new data, the random forest scores the highest hit rates for the original and the rebalanced data sets and among the top three for the reduced data sets. The more advanced methods, such as the neural net and the support vector machines, do not perform as well as expected on the data set at hand. Especially the support vector machines are performing below expectations. This suggests that the three different kernel functions chosen are not good choices for this particular data set. As for the neural net, the result may be due to the difficulties of choosing the grid for the net. In this report neural nets with only one hidden layer has been taken into consideration, and that layer only consisted of numbers that could be divided by 5, up to 120 for the insurance data and 50 for the credit data. A larger net with multiple layers may generate a more accurate model but finding the optimal net for the particular data will be very computationally expensive. The generalized linear model performs surprisingly well on this data set, suggesting that there actually is some noticeable linearity in the data set.

7.2 Discussion

We used the generalized linear model as a standard statistical example of how the classification would be modelled. When comparing this model to the machine learning models, we see that it outperforms some but is inferior to some. When looking at the AUC of the different models we see that we would have chosen the random forest for unaltered data or the SVM-linear kernel for 50% SMOTE as the best model to predict new customers. When comparing the hit rates of these models to the GLM, the random forest performs better but the SVM does not. When using the best threshold to measure hit rate, these would not be the best models. We instead get the best results for random forest and SVM radial for 70% SMOTE and the models we chose would perform much worse. Our data set is very skewed and converges to a hit rate of 0.0595 for all 4000 data points. Therefore the hit rates differ a lot depending on how many samples are chosen. If we classify correctly the probability of choosing a correct point decreases more than it increases when misclassifying. This can be seen in Figure 4.1. The SVM-radial only chose 45 points and therefore the hit rate is very high when nine of those were correct. When looking at the random forest for the 70% SMOTE, 224 points are chosen and the 20.5 % hit rate is achieved. This is a much better result since we chose 5 times as many points and even got a slightly better hit rate. When going left in Table 3 the random forest models increase the chosen number of data points and the hit rate decreases but it is still above all the other model's hit rates. The conclusion here is that the random forest for unaltered data outperforms all the models choosing a lower amount of data points than 300 points. When comparing the models that chooses a higher amount of data points, the GLM and the SVM with linear kernel for 30% SMOTE would be the best performing models, but it is hard to compare with the models choosing lower amount of data points since the hit rate is lower, as expected for a larger set of points. This can be seen in Figure 4.1.

The previous discussion regards our best chosen threshold according to the true positive-false positive relationship. The case that our best model would have the best threshold that matches the problem we are trying to solve is highly unlikely. The random forest for unaltered data chooses 1012 points, if the problem was to choose 1012 points it would perform well but that might not be the case. Instead when choosing the model, it is best to make the decision based on how many points the problem needs. The best thresholds are not always at these points and it could be valuable to analyze the ROC curves. If the ROC curve is very steep in the beginning, it will perform well on a small amount of points and if it has a best threshold very high in the curve this could be the best model to chose if many points are needed. If the problem does not have any requirements, then the pure AUC number would be the best performance measure.

When constructing the machine learning models, we have applied a straight forward grid search to find the parameters. This is a rather crude and inefficient approach, especially for the more complex models whose parameters in theory have no real boundaries. It is likely that an approach using optimization techniques to retrieve the model parameters would not only be more efficient but also generate more accurate models. Since the choice of kernel function for the support vector machines is of utmost importance, it is likely that a different kernel function would be a better fit for the data set than the three we have tried in this report. The difficulty of finding the best model parameters is especially clear for the neural network as it is extremely hard to analyze how the method operates in its nodes. Here we have chosen to create neural nets with only one hidden layer and a sparse number of nodes, and they perform in line with the other models. There is no certainty that a more complex net with multiple hidden layers would improve the performance, but is likely since we have only tried a small fragment of all possible net combinations. There is currently a lot of research being done on neural networks, and we see a lot of potential for the method to outperform other methods within the field of customer relations and risk management.

Further improvements can be achieved by preprocessing the data by rebalancing the ratio of the target class which we here have done using the SMOTE. For our predictions this mainly improved the performance for the support vector machines, especially with the linear kernel function. The other models show similar (or worse) performance on the SMOTE data which suggests that they handle the skewed data well. Even though the SMOTE did not improve our results substantially, it is a valuable tool to use for data more skewed than ours. Reducing the dimensionality of the input is an additional way of improving the efficiency of the model training as well as the performance of the model. Here we used the 15 features that were most important according to the generalized linear model. Once again we did not see any particular improvement the original data set and neither on the SMOTE data sets. Using another reduction approach, such as principal component analysis which selects the features that contribute with the highest variance, may be a better choice. The reduction can also be done by first applying unsupervised learning methods to identify clusters within the data and apply supervised learning techniques on to these clusters.

The credit data set showed that the trend is that the models picking a low number of data points get the highest hit rates. Since this data set is also slightly skewed this is expected, just as we saw for the insurance data. The random forest and the neural network get the highest AUCs but they choose many points according to the best threshold and therefore get much lower hit rate. Therefore the GLM gets the best results. The SMOTE versions also gets much better hit rates than the unaltered ones even though they all have worse AUC values, which would cause problems when applying this on a real problem. The k-NN SMOTE version performs much worse than for the unaltered data, which is probably because SMOTE oversampling is based on k-NN and that these new points are just helping to overfit the model to the training set.

As for all applications of data mining, efficient computational time is of high priority.

Even though this report has not explicitly taken computational time into account, it has had a big part in the process of generating the model. As with most models, there is a balance between the performance of the model and how much computational time you can afford. We see in our results that the less complex methods, such as naïve Bayes and k-nearest neighbors, show similar performance as the more complex methods, such as neural nets and support vector machines, but their computational time is significantly shorter. The computational time for our best performing method, random forest, is slightly higher than the simplest methods but not even comparable to the more complex ones. With respect to the overall performance over all the different data sets, the random forests computational time is very small and this consolidates its position as the most suitable method for this type data and problem. The neural network and the support vector machines show promising results, which suggest that wider net or a different kernel function may perform better, they are too computationally expensive to recommend based on the results in this report alone. In a broader financial perspective, the less complicated methods has the advantage of being easy to interpret, which may be important in order to explain the process to clients or for juridical restrictions which imposes transparency. In the extension, machine learning methods can be applied to risk valuation and other fields that have much more strict regulatory requirements.

Chapter 8

Further work

Machine learning is applicable within many fields. There are several potential extensions this report can have, both deeper and wider. There is much to be done within the models we have chosen. One could focus on computational costs and how they affect the work and how you can cut these using optimization methods.

One could go deeper into exactly how the methods classify and, for instance, use different kernel functions for the support vector machines than we did.

One could manipulate data further than our research. By using different sampling methods, component analysis and feature reduction. Other problems like missing data could also be encountered, but our research does not include this problem.

More statistics to better validate the methods could be of use. Especially when trying to model risk for regulatory purposes, one must have a way to show the regulators that the model works.

From a insurance business perspective, it would be of great interest to not just target the customers that are most likely to purchase an insurance, but out of these be able to target those who are least risky. A potential way to do this is combining machine learning methods with copulas.

Bibliography

Anderson, E. (1936), 'The species problem in iris', *Annals of the Missouri Botanical Garden* **23**(3), 457–509.

Bishop, C. M. (2006), *Pattern recognition and machine learning.*, Information science and statistics, New York, NY : Springer, cop. 2006.

Breiman, L. (1996), 'Bagging predictors.', *Machine Learning* **24**(2), 123.

Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P. (2011), 'Smote: Synthetic minority over-sampling technique.'.

Fawcett, T. (2006), 'An introduction to roc analysis.', *Pattern Recognition Letters* **27**(ROC Analysis in Pattern Recognition), 861 – 874.

Fisher, R. A. (1936), 'The use of multiple measurements in taxonomic problems.', *Annals of Eugenics* **7**(2), 179–188.

from Jed Wing, M. K. C., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., Scrucca, L., Tang, Y., Candan, C. and Hunt., T. (2016), *caret: Classification and Regression Training.* R package version 6.0-73.
URL: *https://CRAN.R-project.org/package=caret*

Hastie, T., Tibshirani, R. and Friedman, J. (2009), *The elements of statistical learning : data mining, inference, and prediction.*, Springer series in statistics, New York : Springer, 2009.

Hill, T. and Lewicki, P. (2007), *STATISTICS: Methods and Applications.*, StatSoft, Tulsa, OK, 2007.
URL: *http://www.statsoft.com/textbook/*

Lewis, D. (1998), *Naive(Bayes)at forty: The independence assumption in information retrieval.*, Vol. 1398 of *Lecture Notes in Computer Science*, Springer Verlag, AT and T Labs.

Lichman, M. (2013), 'UCI machine learning repository'.
URL: *http://archive.ics.uci.edu/ml*

Murphy, K. P. (2012), *Machine learning : a probabilistic perspective.*, Adaptive computation and machine learning series, Cambridge, MA : MIT Press, cop. 2012.

R Core Team (2016), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.
URL: *https://www.R-project.org*

Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J.-C. and Müller, M. (2011), 'proc: an open-source package for r and s+ to analyze and compare roc curves', *BMC Bioinformatics* **12**, 77.

Torgo, L. (2010), *Data Mining with R, learning with case studies*, Chapman and Hall/CRC.
URL: *http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR*

van der Putten, P. and van Someren., M. (2000), 'Coil challenge 2000: The insurance company case.', `https://archive.ics.uci.edu/ml/datasets/Insurance+Company+Benchmark+(COIL+2000)`. Accessed: 2017-02-15.

Yeh, I.-C. and Lien, C.-h. (2009), 'The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients.', `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients#`. Accessed: 2017-03-24.

Youden, W. J. (1950), 'Index for rating diagnostic tests.', *Cancer (0008543X)* **3**(1), 32.

Appendices

Appendix A

Insurance data

DATA DICTIONARY

Nr Name Description Domain
1 MOSTYPE Customer Subtype see L0
2 MAANTHUI Number of houses 1 − 10
3 MGEMOMV Avg size household 1 − 6
4 MGEMLEEF Avg age see L1
5 MOSHOOFD Customer main type see L2
6 MGODRK Roman catholic see L3
7 MGODPR Protestant ...
8 MGODOV Other religion
9 MGODGE No religion
10 MRELGE Married
11 MRELSA Living together
12 MRELOV Other relation
13 MFALLEEN Singles
14 MFGEKIND Household without children
15 MFWEKIND Household with children
16 MOPLHOOG High level education
17 MOPLMIDD Medium level education
18 MOPLLAAG Lower level education
19 MBERHOOG High status
20 MBERZELF Entrepreneur
21 MBERBOER Farmer
22 MBERMIDD Middle management
23 MBERARBG Skilled labourers
24 MBERARBO Unskilled labourers
25 MSKA Social class A

```
26 MSKB1 Social class B1
27 MSKB2 Social class B2
28 MSKC Social class C
29 MSKD Social class D
30 MHHUUR Rented house
31 MHKOOP Home owners
32 MAUT1 1 car
33 MAUT2 2 cars
34 MAUT0 No car
35 MZFONDS National Health Service
36 MZPART Private health insurance
37 MINKM30 Income < 30.000
38 MINK3045 Income 30−45.000
39 MINK4575 Income 45−75.000
40 MINK7512 Income 75−122.000
41 MINK123M Income >123.000
42 MINKGEM Average income
43 MKOOPKLA Purchasing power class
44 PWAPART Contribution private third party insurance see L4
45 PWABEDR Contribution third party insurance (firms) ...
46 PWALAND Contribution third party insurance (agriculture)
47 PPERSAUT Contribution car policies
48 PBESAUT Contribution delivery van policies
49 PMOTSCO Contribution motorcycle/scooter policies
50 PVRAAUT Contribution lorry policies
51 PAANHANG Contribution trailer policies
52 PTRACTOR Contribution tractor policies
53 PWERKT Contribution agricultural machines policies
54 PBROM Contribution moped policies
55 PLEVEN Contribution life insurances
56 PPERSONG Contribution private accident insurance policies
57 PGEZONG Contribution family accidents insurance policies
58 PWAOREG Contribution disability insurance policies
59 PBRAND Contribution fire policies
60 PZEILPL Contribution surfboard policies
61 PPLEZIER Contribution boat policies
62 PFIETS Contribution bicycle policies
63 PINBOED Contribution property insurance policies
64 PBYSTAND Contribution social security insurance policies
65 AWAPART Number of private third party insurance 1 − 12
66 AWABEDR Number of third party insurance (firms) ...
```

67 AWALAND Number of third party insurance (agriculture)
68 APERSAUT Number of car policies
69 ABESAUT Number of delivery van policies
70 AMOTSCO Number of motorcycle/scooter policies
71 AVRAAUT Number of lorry policies
72 AAANHANG Number of trailer policies
73 ATRACTOR Number of tractor policies
74 AWERKT Number of agricultural machines policies
75 ABROM Number of moped policies
76 ALEVEN Number of life insurances
77 APERSONG Number of private accident insurance policies
78 AGEZONG Number of family accidents insurance policies
79 AWAOREG Number of disability insurance policies
80 ABRAND Number of fire policies
81 AZEILPL Number of surfboard policies
82 APLEZIER Number of boat policies
83 AFIETS Number of bicycle policies
84 AINBOED Number of property insurance policies
85 ABYSTAND Number of social security insurance policies
86 CARAVAN Number of mobile home policies 0 − 1

L0:

Value	Label
1	High Income, expensive child
2	Very Important Provincials
3	High status seniors
4	Affluent senior apartments
5	Mixed seniors
6	Career and childcare
7	Dinki's (double income no kids)
8	Middle class families
9	Modern, complete families
10	Stable family
11	Family starters
12	Affluent young families
13	Young all American family
14	Junior cosmopolitan
15	Senior cosmopolitans
16	Students in apartments

```
17 Fresh masters in the city
18 Single youth
19 Suburban youth
20 Ethnically diverse
21 Young urban have−nots
22 Mixed apartment dwellers
23 Young and rising
24 Young, low educated
25 Young seniors in the city
26 Own home elderly
27 Seniors in apartments
28 Residential elderly
29 Porchless seniors: no front yard
30 Religious elderly singles
31 Low income Catholics
32 Mixed seniors
33 Lower class large families
34 Large family, employed child
35 Village families
36 Couples with teens 'Married with children'
37 Mixed small town dwellers
38 Traditional families
39 Large religious families
40 Large family farms
41 Mixed rurals


L1:

1 20−30 years
2 30−40 years
3 40−50 years
4 50−60 years
5 60−70 years
6 70−80 years


L2:

1 Successful hedonists
2 Driven Growers
```

3	Average Family
4	Career Loners
5	Living well
6	Cruising Seniors
7	Retired and Religious
8	Family with grown ups
9	Conservative families
10	Farmers

L3:

0	0%
1	1 – 10%
2	11 – 23%
3	24 – 36%
4	37 – 49%
5	50 – 62%
6	63 – 75%
7	76 – 88%
8	89 – 99%
9	100%

L4:

0	f 0
1	f 1 – 49
2	f 50 – 99
3	f 100 – 199
4	f 200 – 499
5	f 500 – 999
6	f 1000 – 4999
7	f 5000 – 9999
8	f 10.000 – 19.999
9	f 20.000 – ?

Appendix B

Reduced features

Feature nr.	p-value
47	$5.886696 \cdot 10^{-6}$
59	$1.213239 \cdot 10^{-2}$
76	$3.124046 \cdot 10^{-2}$
82	$6.082598 \cdot 10^{-2}$
57	$8.859511 \cdot 10^{-2}$
44	$9.514496 \cdot 10^{-2}$
80	$9.687050 \cdot 10^{-2}$
43	$9.879094 \cdot 10^{-2}$
22	$1.000583 \cdot 10^{-1}$
4	$1.313617 \cdot 10^{-1}$
78	$1.380787 \cdot 10^{-1}$
33	$1.403834 \cdot 10^{-1}$
32	$1.540714 \cdot 10^{-1}$
38	$1.650137 \cdot 10^{-1}$
40	$1.667806 \cdot 10^{-1}$

(a) Unaltered data.

Feature nr.	p-value
4	$1.952570 \cdot 10^{-8}$
57	$3.480886 \cdot 10^{-7}$
78	$1.300638 \cdot 10^{-6}$
85	$2.691749 \cdot 10^{-5}$
47	$5.169883 \cdot 10^{-5}$
59	$4.177716 \cdot 10^{-4}$
82	$5.770738 \cdot 10^{-4}$
64	$1.453220 \cdot 10^{-3}$
55	$2.794721 \cdot 10^{-3}$
27	$2.927923 \cdot 10^{-3}$
76	$3.617026 \cdot 10^{-3}$
29	$4.732140 \cdot 10^{-3}$
6	$7.521170 \cdot 10^{-3}$
38	$1.045566 \cdot 10^{-2}$
18	$1.228985 \cdot 10^{-2}$

(b) 30% SMOTE data.

Table B.1: Top 15 features with lowest p-values using GLM. The marked feature numbers are the ones which belong to the top 15 in every data set.

Feature nr.	p-value
85	$2.533193 \cdot 10^{-10}$
70	$2.899989 \cdot 10^{-9}$
64	$2.996851 \cdot 10^{-8}$
49	$8.793468 \cdot 10^{-8}$
4	$5.493286 \cdot 10^{-7}$
57	$8.889321 \cdot 10^{-6}$
78	$3.499542 \cdot 10^{-5}$
59	$3.499542 \cdot 10^{-5}$
65	$8.374254 \cdot 10^{-4}$
76	$8.851775 \cdot 10^{-4}$
32	$9.802717 \cdot 10^{-4}$
27	$1.069206 \cdot 10^{-3}$
68	$1.431689 \cdot 10^{-3}$
25	$2.777576 \cdot 10^{-3}$
33	$2.818616 \cdot 10^{-3}$

(a) 50% SMOTE data.

Feature nr.	p-value
85	$1.382454 \cdot 10^{-8}$
4	$9.867834 \cdot 10^{-8}$
64	$5.167611 \cdot 10^{-7}$
59	$1.829774 \cdot 10^{-6}$
24	$3.983315 \cdot 10^{-6}$
29	$5.245012 \cdot 10^{-6}$
70	$6.180093 \cdot 10^{-6}$
80	$6.286635 \cdot 10^{-6}$
22	$1.789698 \cdot 10^{-5}$
65	$2.010815 \cdot 10^{-5}$
27	$2.248124 \cdot 10^{-5}$
49	$3.324108 \cdot 10^{-5}$
23	$3.430147 \cdot 10^{-5}$
44	$4.979897 \cdot 10^{-4}$
32	$9.085402 \cdot 10^{-4}$

(b) 70% SMOTE data.

Table B.2: Top 15 features with lowest p-values using GLM. The marked feature numbers are the ones which belong to the top 15 in every data set.

Appendix C

Credit data

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:

X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.

X2: Gender (1 = male; 2 = female).

X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).

X4: Marital status (1 = married; 2 = single; 3 = others).

X5: Age (year).

X6 – X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .; X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

X12–X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.

X18–X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .; X23 = amount paid in April, 2005.

Master's Theses in Mathematical Sciences 2017:E23
ISSN 1404-6342

LUTFMS-3320-2017

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/