High-risk Consumer Credit Scoring using Machine Learning Classification

Ludvig Levay
ludvig.levay@gmail.com

Max Mjörnell
max.mjornell@gmail.com

Supervisor: Magnus Wiktorsson

Examiner: Johan Lindström

Department of Mathematical Statistics
Lund University, Faculty of Engineering

May 14, 2019

Printed in Sweden
Matematikhuset, Lund

Abstract

The use of statistical models in credit rating and application scorecard modelling is a thoroughly explored field within the financial sector and a central component in a credit institution's underlying business model. The aim of this report was to apply and compare six different machine learning models in predicting credit defaults for high-risk consumer credits, using a data set provided by a Swedish consumer credit institute. The selected models include the ones most frequently used for scorecard modelling across the banking industry as well as some more rarely used that could potentially add valuable insights. The models are briefly introduced and the most important concepts for each model are explained, as well as how to deal with the lack of transparency in complex models by the use of white-boxing methods. Appropriate metrics for evaluating prediction performance on imbalanced and insufficient data sets are discussed, as well as how to increase model performance by using different oversampling techniques. All available information about the loan applicants was then exhaustively examined, and a carefully refined set of input features was constructed to ensure optimal predictive power and generalizability. After tuning and testing the models, the results showed that logistic regression, support vector machine, neural network and a soft voting ensemble showed similar performance results using the same input feature configurations. Attempts to create synthesized samples to handle the imbalance problem showed no effect and was therefore not used. The white-boxing model SHAP showed a promising ability to instructively explain the underlying decision basis for complex models such as neural networks. However, considering the limited data set at hand, the recommended model to use is the logistic regression model given its simplicity and on a par performance with the other models. Having larger amounts of data available on the other hand, the more complex models such as neural networks and support vector machines could have a potential advantage.

Keywords:
Machine learning, Scorecard modelling, Logistic regression, Support Vector Machine, Decision Tree, Random Forest, k-Nearest Neighbors, Artificial Neural Network, Voting ensemble, SHAP, LIME, Average Precision score, Feature engineering

Popular Science Summary

Machine learning has recently become the word on everyone's lips, and its application areas are currently booming. Financial institutions are frequently using new discoveries to automate and optimize their businesses, including the issue of rating loan applicants' creditworthiness. The most widely used approach today is either using a simple statistical model or judgmental decision-making, where credit analysts use prior knowledge to predict whether a loan will default or not. In this report, some of the most popular machine learning algorithms are successfully applied on the area of application scorecard modelling. The aim is to investigate if cutting-edge models could outperform more outdated approaches such as logistic regression on a relatively small data set to gain knowledge from. In addition, techniques to increase human interaction was implemented in order for a human to understand complex machine learning classifications. The results showed that complex models such as an artificial neural network did not top the simpler models, and the reason is believed to be the limited amount of available data. Given more data on the other hand, the more sophisticated machine learning classifiers are believed to have a prosperous future in credit scoring. They were also successfully deciphered, so a human easily could interpret even the most complicated operation. The resulting models have the potential to significantly decrease default rates if implemented, and in the long run lead to increased profitability as well as fewer people ending up in mountains of debt they can't repay.

Table of Contents

List of Figures

List of Tables

Chapter 1
Background

1.1 Introduction to the problem

Today there are numerous machine learning methods used on a wide variety of application areas, and the use of these techniques is increasing across various industries. Among banks and credit institutions, credit scoring has been used for more than 50 years to assess risk. The first model was proposed in US in 1941, and credit scoring received complete acceptance in 1975, with the passing of the *US Equal Credit Opportunity Act I*. A credit scoring model assigns scores to the borrower based on characteristics of the borrower as well as other factors such as historical default that may indicate a higher risk. Prior to these more formal methods, judgmental decisions were made by the credit analyst based on personal knowledge of the applicant. This method has many shortcomings since it is unreliable, not replicable, hard to teach, not scalable and subjective. To be able to determine a loan applicant's creditworthiness, the credit institutions gather data about the applicant, including financial, demographic and other relevant information. This is however a difficult classification problem since it is hard to find characteristics that have enough discriminatory power to make an accurate scoring. However, a reliable scoring model can provide a helpful tool for credit analysts, assisting them with statistical facts based on previous customers. (Bolton, 2009, p.8-9)

In this report, six different machine learning methods are compared to predict default risk for customers to a Swedish consumer credit institute. The models that are used are logistic regression, support vector machine, tree-based classifiers (including decision tree and random forest), k-nearest neighbors, artificial neural networks and voting models. They are considered to cover the most widely used techniques within credit rating today as well as some rather uncommon for these types of classification problems. Furthermore, the models have different advantages and disadvantages when it comes to complexity, transparency, computational power and stability.

1.2 Problem description

The data set for this report was provided by a Swedish consumer credit institute **(hereafter referred to as "the Company")**, and consists of 8771 unique loan applications between June 2017 - August 2018. The product is a high-risk consumer credit, which generally is a smaller loan between 1-25 kSEK with higher interest rates and shorter duration compared to e.g. mortgages or car financing loans. Also, the loan is without security, meaning that the customer does not need a security such as a property to be granted a loan. According to the Swedish consumer credit law, a high-risk (or high-cost) consumer credit is defined as a loan to a private consumer with an interest rate higher than the Swedish reference rate plus 30 percent, and that is not tied to a purchase of a specific product or property (Swedish Code Of Statutes, 2018, 2018:478 2§). The data set consists of both new customers applying for a credit as well as recurring customers that want to extend their current credit limit.

The outcomes are denoted "good" and "bad", which is determined by the loan's condition after 6 months from that the application was granted. If the customer has failed to follow the repayment plan after 6 months, the outcome is categorized as "bad", otherwise "good". About 14.8% of the data is categorized as "bad" loans. Ideally, one would have wanted to evaluate the loan condition after at least 12 months from that the loan was granted. However, given that data only was available approximately 18 months back in time, that would have resulted in a significant loss of data samples to use. Given the relatively short duration of the loans (between 1-12 months), it was regarded sufficient with 6 months as evaluation time.

The aim of this report is to provide a statistical model that can be used to support the Company's credit analysts when making credit assessments. One challenge is the combination of an imbalanced data set and a limited amount of historical data. Another issue is stability, since the models are expected to be used over time and must handle changing characteristics for new customers. To deal with the lack of data and imbalance between defaulting and non-defaulting customers, various approaches of over-sampling techniques will be compared and used to determine whether it makes an improvement or not. To assess the stability issue, appropriate metrics are used as well as cross-validation to combat variance of performance metrics. The resulting models should, to the degree possible, meet following criteria:

- Precise and efficient, as being able to prevent a large share of potentially defaulting customers to be granted a loan whilst maintaining a high precision to avoid rejecting too many non-defaulting ("good") customers.
- Risk agile, so that the Company can monitor the trade-off between precision and how restrictive they want to be on a monthly or weekly basis.
- Generalizable and stable over time, so that it can be trusted over a given period of time in the future without hurting performance significantly.
- Interpretable to the user, so that every scoring made can be decomposed and understood by a human.

1.3 Limitations

This study only uses data from loan applications that have been granted by the Company and how they have performed. Thus, the models will not be able to evaluate potential customer segments that the Company rejects today. This creates a bias where information about rejected customers is excluded from the modelling. To reduce this bias, one could take into account the rejected credit applications and create a proxy on their performance based on publicly available information. However, since the Company erases information about non-active customers within three months in accordance with GDPR, this was not possible for this study.

Furthermore, the outcome of the loans is denoted "good" or "bad", i.e. a binary outcome is assumed. Ideally, several risk categories depending on level of risk for each loan should be used to get a more nuanced perception of a loan's potential default risk and recovery rates in case of default. A clear distinction between different risk segments other than "good" or "bad" was however not available within this data set. Additionally, given the increased complexity in multiclass classification and the nature of the data set at hand, the chosen approach was to limit the problem to binary classification.

External factors in order to handle unlikely macroeconomic events such as a financial crisis are generally included by credit institutions for credit ratings when assessing mortgage risks and company valuations. The effect of external factors is excluded in this report, given the short loan duration and relatively small loan amounts. The models are only intended to be used in relatively stable economic circumstances and should be discarded if e.g. a severe economic recession occurs.

Chapter 2
Classifiers

2.1 Logistic regression

Logistic regression is considered the most favored credit scoring method for practical use in the banking industry (Bolton, 2009, p.19). There are two main differences in assumptions between an ordinal linear regression and logistic regression. To generalize linear regression to the case when the output is binary, $y \in \{0, 1\}$, the Gaussian distribution of y is replaced by a Bernoulli distribution, i.e.

$$p(y|x, w) = Ber\Big(y|\mu(x)\Big) \tag{2.1}$$

Where x is the input and $\mu(x) = E[y|\boldsymbol{x}] = p(y = 1|x)$. Furthermore, one has to ensure that $0 \leq \mu(x) \leq 1$. This is done by computing a linear combination of the inputs, and then mapping the sum through the sigmoid function, also known as logit function. $\mu(x)$ is now defined as

$$\mu(x) = sigm(w^T x) = \frac{1}{1 + e^{-w^T x}} = \frac{e^{w^T x}}{e^{w^T x} + 1} \tag{2.2}$$

To determine the model parameters, the Negative log-likelihood is minimized. The NLL is defined as

$$NLL(\boldsymbol{w}) = -\sum_{i=1}^{N} log\Big(\mu_i^{\mathbb{1}(y_i=1)} \times (1 - \mu_i)^{\mathbb{1}(y_i=0)}\Big) =$$

$$= -\sum_{i=1}^{N} \Big(y_i log\mu_i + (1 - y_i) log(1 - \mu_i)\Big) \tag{2.3}$$

Since the problem is not on closed form as in the case of linear regression, one has to use an optimization algorithm to compute the maximum likelihood estimators. Common used methods are steepest descent, Newton's method and quasi-Newton methods. (Murphy, 2006, p.22, 250-252)

2.2 Tree-based classifiers

2.2.1 Decision tree

A decision tree is a function that returns a single output value, or a "decision", taking a vector of attribute values as input. In a Boolean decision tree, there are exactly two possible output values. The architecture of the model is represented as a tree with internal nodes, where each internal node in the tree corresponds to a test of the value of one of the input attributes. The tree's branches are the paths created depending on the outcome of the tests at each node.
Decision trees is a simple but widely used model in scorecard modelling. It works by recursively splitting the input space, and defining a simple local model in each resulting region of input space. The classification procedure for a new input x can be illustrated as a binary tree as in Figure 2.1.

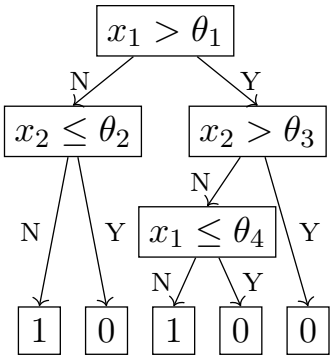

Figure 2.1: Decision tree

In the first node (the root), the input space is divided into two regions depending on which side of the threshold θ_1 that x_1 belongs, where θ_1 is a parameter of the decision tree model. These two subregions can then be subdivided independently a finite number of steps. The nodes on the last layer of the tree (the leaves) are the final subregions in which the inputs are categorized. To determine the classification of any new input x, you start at the root of the tree and follow a path down to a specific leaf according to the specified decision criteria at each passed node. Each region is then assigned a specific class, in this example 1 or 0, where the classification is depending on the class distribution in every leaf. (Bishop, 2006, p.663-664)

The decision tree model can be written on the following form:

$$f(\boldsymbol{x}) = E[y|\boldsymbol{x}] = \sum_{m=1}^{M} w_m \mathbb{1}(\boldsymbol{x} \in R_m) = \sum_{m=1}^{M} w_m \phi(\boldsymbol{x}; v_m) \tag{2.4}$$

Where R_m is the *m'th* region, w_m the mean response (or class distribution) in region m, and v_m the choice of variable to split on, and the threshold value on the path from the first node to the *m'th* leaf. Since the optimal solution is NP-complete, a greedy procedure is commonly used to compute a locally optimal maximum likelihood estimation. The function that measures if a node is worth splitting is defined as

$$\triangle cost(D) - \sum_{i=1}^{n} \frac{|D_i|}{|D|} cost(D_i) \tag{2.5}$$

Where D is the data in the current leaf, and D_i is the new amount of data in leaf i after splitting D into n new nodes.

Before defining the cost functions, the estimations of class-conditional probabilities for the data D in a leaf is calculated as:

$$\hat{\pi}_c = \frac{1}{|D|} \sum_{i \in D} \mathbb{1}(y_i = c) \tag{2.6}$$

Where $\hat{\pi}_c$ is the probability that a random entry in the leaf belongs to class c. There are three mainly used cost functions in classification problems that measure quality of a split. Those are:

- Misclassification rate. The most probable class label is defined as $\hat{y}_c = \text{argmin}_c \, \hat{\pi}_c$, with the corresponding error rate
$$1 - \hat{\pi}_{\hat{y}} = \frac{1}{|D|} \sum_{i \in D} \mathbb{1}(y_i \neq \hat{y}) \tag{2.7}$$
- Entropy, or deviance
$$H(\hat{\boldsymbol{\pi}}) = -\sum_{c=1}^{C} \hat{\pi}_c log(\hat{\pi}_c) \tag{2.8}$$
- Gini index, or the expected error rate
$$\sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c) = \sum_{c=1}^{C} \hat{\pi}_c - \sum_{c=1}^{C} \hat{\pi}_c^2 = 1 - \sum_{c=1}^{C} \hat{\pi}_c^2 \tag{2.9}$$

Growing an optimal (and thereby increasingly complex) tree will eventually lead to overfitting. The approach of stop growing the tree by examining if the decrease in error justifies the increased complexity is usually not a successful method. If the discriminatory power in each feature on its own is too low, it would eventually make no or very few splits. Instead, a method called *pruning* is performed. This means that a "full" tree is grown, where the branches are removed one at a time in a way that results in the lowest increase in error. (Murphy, 2006, p.544-546)

2.2.2 Random forest

Another approach to solve the common problem with overfitting when using decision tree classification is a method called random forest. The model splits the training data into M different subsets, and trains M different trees on each of the subsets, and then computes the ensemble

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \frac{1}{M} f_m(\boldsymbol{x}) \tag{2.10}$$

Where f_m is the *m'th* tree and M the number of trees in the ensemble. This technique is commonly known as bagging or "bootstrap aggregating". There is an issue though. When using the same algorithm multiple times on different subsets of the training data, the resulting trees can be highly correlated, which increases the prediction variance. By picking a randomly chosen subset of input variables as well as randomly picking subsets of data cases, this effect can be dealt with. (Murphy, 2006, p.550-551)

2.3 k-Nearest Neighbors

The k-Nearest Neighbors (k-NN) model is one of the simplest machine learning algorithms. The method assesses the similarities between the pattern identified in the training set and the input pattern. A new data point will be assigned to the respective class c of which the majority of the neighbors belong by considering the k points in the training set that are nearest to the test input x. In this case, "nearest" refers to the training set samples that are most similar to the observed data point when looking at input feature values in x. This can be expressed as

$$p(y = c|x, D, k) = \frac{1}{k} \sum_{i \in N_k(x,D)} \mathbb{I}(y_i = c) \tag{2.11}$$

Where $N_K(x, D)$ are the indices of the k nearest points to x in the training set D. When performing k-NN, a crucial part is choosing the metric to use. A commonly used metric is the standard Euclidean norm given by

$$p_1(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})'(\boldsymbol{x} - \boldsymbol{y})} \tag{2.12}$$

Where $\boldsymbol{x}$ and $\boldsymbol{y}$ are measurement vectors. When choosing the k, i.e. the number of nearest neighbors to consider, there is a trade-off between variance and flexible fitting. Generally, a lower k allows more complex classification but with the cost of a higher variance, since it will be more affected by potential outliers and irregularity in the training data. The higher the k, the more samples are used in the classification and hence a more stable model, with the drawback of being less flexible and precise. (Murphy, 2006, p.16-17)

2.4 Support vector machine

Support vector machine (or SVM) is a classification algorithm (commonly referred to as a support vector classifier in the case of classification problems) that computes the optimal separating hyperplane for linearly separable patterns. SVMs are also capable of classifying non-linear patterns by transforming the original data to map into higher dimensions where the data is linearly separable. Compared to other classification algorithms, SVM considers only the data points closest to the decision hyperplane, i.e. the data points most difficult to classify, to determine the optimal hyperplane. These data points are referred to as support vectors. In Figure 2.2 an illustration of a separating hyperplane in two dimensions is shown, where the support vectors are marked in red:

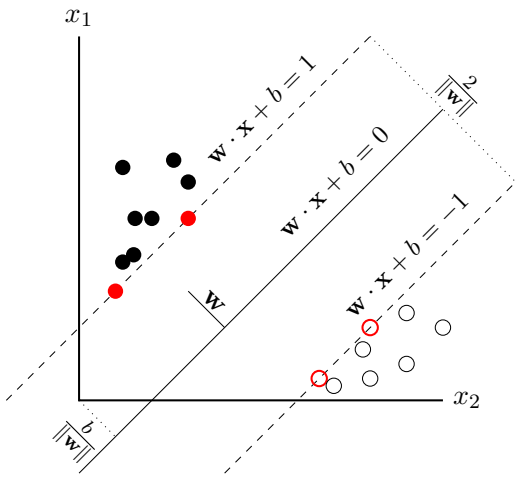

Figure 2.2: Support vector machine

The hyperplanes H are defined such that:

$$wx_i + b \geq +1 \text{ when } y_i = +1$$
$$wx_i + b \leq -1 \text{ when } y_i = -1$$

Where y is the target value, w is a weight vector, x the input vector and b is a bias.

H_1 and H_2 are the planes:

$$H_1\text{: } wx_i + b = +1$$
$$H_2\text{: } wx_i + b = -1$$

The plane H_0 is the median of H_1 and H_2. $d+$ denotes the shortest distance to the closest positive point and d- the shortest distance to the closest negative point, which defines the margin of the separating hyperplane. The optimal hyperplane is then given by the hyperplane that maximizes the margin of separation d. The classifications of new data points are then determined by looking at the sign of the function

$$f(x) = wx + b \tag{2.13}$$

In the case of a linear separator, the distance d is maximized. The distance between H_0 and H_1 can be expressed as

$$d = ||w \cdot x_i + b|| = \frac{1}{||w||} \tag{2.14}$$

The total distance between H_1 and H_2 thus becomes $2/||w||$. This means that the margin is maximized by minimizing

$$\frac{1}{2}||w|| \tag{2.15}$$

with the condition that no data points are between H_1 and H_2, i.e.

$$\begin{cases} \mathrm{x}_i \cdot w + b \geq 1 \text{ when } y_i = 1 \\ \mathrm{x}_i \cdot w + b \leq -1 \text{ when } y_i = -1 \end{cases}$$

which can be combined into following constraint:

$$y_i(x_i \cdot w + b) \geq 1 \tag{2.16}$$

This is now a constrained optimization problem that can be solved using the Lagrange function with the constraints in (2.15) and (2.16):

$$L(w, b) = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} a_i[y_i(wx_i + b) - 1] \tag{2.17}$$

s.t. $\forall i, a_i \leq 0$ where a_i is the Lagrangian multiplier and N is the number of data points in the training set. From calculating the derivatives of (2.17) with respect to w and b and put the derivatives equal to 0 yields

$$\frac{\partial L}{\partial w} = 0 \Leftrightarrow w = \sum_{i=1}^{N} a_i y_i x_i \tag{2.18}$$

$$\frac{\partial L}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^{N} a_i y_i = 0 \tag{2.19}$$

Instead of minimizing over w, b, subject to constraints involving a, the Lagrangian dual problem instead maximizes over a subject to the relations stated for w and b in (2.18) and (2.19) respectively. Substituting (2.18) and (2.19) in (2.17), the dependency of w and b is removed, and results in the Lagrangian dual equation:

$$\hat{L}(a) = \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j \Big(x_i \cdot x_j \Big) \tag{2.20}$$

Subject to

$$\begin{cases} \sum_{i=1}^{N} a_i y_i = 0 \\ a_i \geq 0 \end{cases}$$

Introducing a so-called Kernel Trick, the SVM can be computed for non-linear decision surfaces. One defines a function $\phi(x)$ that maps the data into a feature space of a higher dimension, and the Lagrangian dual equation in (2.20) instead becomes

$$\hat{L}(a) = \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j \Big(\phi(x_i) \cdot \phi(x_j) \Big) \tag{2.21}$$

Depending on the polynomial grade of the function $\phi(x)$, calculating the dot product will be computationally expensive. Instead, defining the kernel function K such that $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$, the dot product computation is not needed since the kernel function defines inner products in the transformed space. Thus, the function to optimize becomes

$$\hat{L}(a) = \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j K \Big(x_i \cdot x_j \Big) \tag{2.22}$$

Subject to

$$\begin{cases} \sum_{i=1}^{N} a_i y_i = 0 \\ a_i \geq 0 \end{cases}$$

(MIT-OpenCourseWare, retrieved 2019)

To classify a new data sample, the sign of $f(x)$ expressed in (2.13) is written in terms of the kernel function and a by substituting w with the expression in (2.18):

$$f(x) = \sum_{i=1}^{N} a_i y_i K\Big(x, x_i\Big) + b \tag{2.23}$$

A constrained optimization of this form satisfies the KKT conditions, which means that following holds:

$$\begin{aligned} a_i &\geq 0 \\ y_i f(x_i) - 1 &\geq 0 \\ a_i\big(y_i f(x_i) - 1\big) &= 0 \end{aligned} \tag{2.24}$$

This means that for every data point in the training set, either $a_i = 0$ or $y_i f(x_i) = 1$. For every data point where $a_i = 0$ will not make a contribution to the sum in (2.23), i.e. will not be involved in classifying new data points. The remaining data points are so called support vectors. Since they fulfill $y_i f(x_i) = 1$, they correspond to data points on the maximum margin hyperplanes. This is a central insight in understanding the use of support vector machines. When training the model, only the support vectors will be retained which significantly decreases the dimensionality of the problem. To determine the parameter b, the fact that any support vector x_i satisfies $y_i f(x_i) = 1$ is used combined with the expression in (2.23)

$$y_i \left(\sum_{j \in S} a_j y_j K\big(x_i, x_j\big) + b \right) = 1 \tag{2.25}$$

where S denotes the set of indices of the support vectors. To solve for b, one multiplies by y_i on both sides, using the fact that $y_i^2 = 1$, and then computing the average over all support vectors:

$$b = \frac{1}{N_S} \sum_{i} \left(y_i - \sum_{j} a_j y_j K\big(x_i, x_j\big) \right) \tag{2.26}$$

where N_S denotes the number of support vectors.

By now, the assumption that the training data is linearly separable in the feature space $\phi(x)$ has been used, meaning that the resulting SVM will give an exact separation. However, in difficult classification problems where there are significant overlaps between class distributions such as in the case of credit scoring, exact separation tends to cause poor generalization. To deal with this, the constraints in (2.16) will be replaced with

$$y_i f(x_i) \leq 1 - \xi_i \tag{2.27}$$

Where ξ_i is a slack variable, that for different values will lead to misclassification:

$$\begin{aligned} \xi_i = 0 &\quad \text{Correctly classified} \\ 0 < \xi_i \leq 1 &\quad \text{Inside the margin} \\ \xi_i > 1 &\quad \text{Misclassified} \end{aligned} \tag{2.28}$$

The function to minimize thus becomes

$$\min C\sum_{i=1}^{N}\xi_i + \frac{1}{2}||w||^2 \tag{2.29}$$

Where C $>$ 0 represents the trade-off between the slack variable penalty and the margin. The responding Lagrange function for minimizing (2.29) with respect to (2.27) becomes

$$L(w,b,a) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}a_i\big(y_i f(x_i) - 1 + \xi_i\big) - \sum_{i=1}^{N}\mu_i\xi_i \tag{2.30}$$

Where $a_i \leq 0$ and $\mu_i \leq 0$ are Lagrange multipliers. The KKT conditions are

$$\begin{aligned} a_i &\geq 0 \\ y_i f(x_i) - 1 + \xi_i &\geq 0 \\ a_i\big(y_i f(x_i) - 1 + \xi_i\big)\big) &= 0 \\ \mu_i &\geq 0 \\ \xi_i &\geq 0 \\ \mu_i\xi_i &= 0 \end{aligned} \tag{2.31}$$

Using the same technique as in (2.20) to eliminate dependency of w, b and ξ_i, the following dual Lagrangian is introduced

$$\hat{L}(\boldsymbol{a}) = \sum_{i=1}^{N}a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}a_i a_j y_i y_j K(x_i, x_j) \tag{2.32}$$

which is minimized with respect to

$$\begin{aligned} 0 \leq a_i \leq C \\ \sum_{i=1}^{N}a_i y_i = 0 \end{aligned} \tag{2.33}$$

As before, the data points where $a_n = 0$ do not contribute to the predictive model, and the remaining points constitute the support vectors, satisfying

$$y_i f(x_i) = 1 - \xi_i \tag{2.34}$$

The parameter b is then estimated by

$$b = \frac{1}{N_M}\sum_{i\in M}\left(y_i - \sum_{j\in S}a_j y_j K(x_i, x_j\right) \tag{2.35}$$

where M represents the set of indices for data points that fulfils $0 < a_i < C$. (Bishop, 2006, p.328-335)

2.5 Artificial neural network

2.5.1 Neural network structures

Feed-forward neural networks, or multilayer perceptrons (MLPs), is a deep learning model with the goal of approximating some function f^*. A feed-forward network defines a mapping $y = f(x; w)$ and learns the value of the parameters w that results in the best function approximation.

Neural networks are composed of nodes connected by directed links. A link from node i to node j serves to propagate the activation from i to j. For each link, there is a corresponding weight $w_{i,j}$ that can be interpreted as the significance of the connection. The feed-forward network is often depicted as a sequence of layers, more specifically an input layer, any number of hidden layers and an output layer. The nodes are constructed in a way that each node receives input only from nodes in the immediately preceding layer. (Russell and Norvig, 2010, p.727-732) In Figure 2.3 a simple feed-forward neural network with one hidden layer is illustrated.

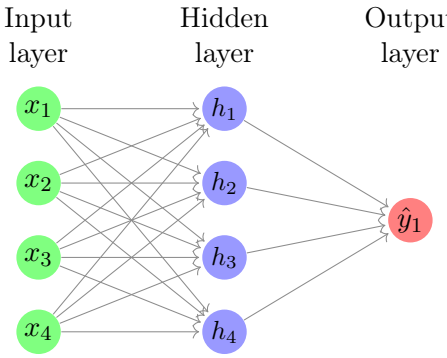

Figure 2.3: Neural network with one input layer, one hidden layer and one output layer

2.5.2 Forward propagation

When using a feed-forward neural network to accept an input x and produce an output $\hat{y}$, information flows forward through the network. The input x provides the initial information that then propagates up to the hidden units at each layer and finally produces $\hat{y}$. This is called *forward propagation*. During training, forward propagation continues onward until it produces an error, which is defined by the error function $E(w)$. (Goodfellow et al., 2016, p.200)
Each node in the feed-forward network computes a weighted sum of its inputs of the form

$$a_j = \sum_i w_{ij} z_i \tag{2.36}$$

Where z_i is the output of node i that sends a connection to unit j, and w_{ij} is the weight associated with that connection from node i to j. The sum of inputs a_j to node j is transformed by the activation function $g(\cdot)$ to give the activation (or output) z_j of node j in the form

$$z_j = g(a_j) \tag{2.37}$$

The corresponding inputs and activations (outputs) for all of the hidden layer and output nodes are calculated for all patterns in the training set, by repeatedly applying (2.36) and (2.37). For a neural network with one input layer, one hidden layer and one output layer, the forward propagation procedure can be formulated as

$$y(x, w) = f\Big(\sum_{j=1}^{M} w_{jk}^{(2)} g\Big(\sum_{i=1}^{D} w_{ij}^{(1)} x_i\Big)\Big) \tag{2.38}$$

Where f is the output activation function and g is the activation function for the hidden layer with $w^{(1)}$ being the trainable weights between input and hidden layer and $w^{(2)}$ being the trainable weights between the hidden layer and the output layer. The variable x is the set of input features acting as the initial z_i. (Bishop, 2006, p.227-240)

2.5.3 Activation functions

Activation functions can be used between layers in a neural network to introduce non-linear mappings to the model. This can enable the model to capture more complex relations in the data.
From the last hidden layer to the output layer it is very common to use an activation function to map the result to something interpretable for the problem at hand. For example, mapping to a number between 0 and 1 can be interpreted as a risk probability, or for a binary problem the activation function could be a step function producing either a 0 or a 1. There are various activation functions used depending on the nature of the data and what the intended use of the model is. The most commonly used activation functions are:

- **Sigmoid functions**

  $$z_j = g(a_j) = \frac{1}{1 + e^{-a_j}} \tag{2.39}$$

  Which maps z_j between 0 and 1. The sigmoidal function is useful since it saturates to 1 when z_j is very positive, saturates to 0 when a_j is very negative, and is only strongly sensitive to the input when z_j is near 0. The logistic sigmoid function has many useful applications, where the perhaps most obvious one is to map a model output to a probabilistic result. It can also be used to approximate step functions between hidden layers, as a step function is not continuously differentiable.

- **Rectified Linear Unit (ReLu)**

  $$z_j = g(a_j) = max\{0, a_j\} \tag{2.40}$$

  The ReLu function essentially removes negative inputs. This may seem illogical but has actually shown to have similarities to observable processes. The ReLu function is commonly used between hidden layers in deep neural networks and is very useful when working with computer vision among other applications.

- **Softmax function**

  $$y_k = f_k(\boldsymbol{z}, \boldsymbol{w}) = \frac{e^{a_k(\boldsymbol{z}, \boldsymbol{w})}}{\sum_j e^{a_j(\boldsymbol{z}, \boldsymbol{w})}} \tag{2.41}$$

  This is a convenient activation function for the output layer in a multiclass classification problem since it both satisfies $0 \leq y_k \leq 1$ as well as $\sum_k y_k = 1$ and thus can be interpreted as probabilities of belonging to each class.

(Bishop, 2006, p.242-245)

2.5.4 Backpropagation

Backpropagation refers to the method of computing the gradient of an error function $E(w)$ for a neural network by propagating errors backwards through the network. This calculation requires high computational power, and is done efficiently using the chain rule of calculus which states that

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} \tag{2.42}$$

The "error" δ is defined as

$$\delta_j = \frac{\partial E_n}{\partial a_j} \tag{2.43}$$

Using (2.36) it can be re-written as

$$\frac{\partial a_j}{\partial w_{ij}} = z_i \tag{2.44}$$

Substituting (2.43) and (2.44) into (2.42) gives the following expression

$$\frac{\partial E_n}{\partial w_{ij}} = \delta_j z_i \tag{2.45}$$

This means that the gradient of the error function with respect to a given weight w_{ij} is computed by multiplying the value of δ for the output end of the weight by the value of z for the unit at the input end of the weight. To find δ for the hidden nodes, the chain rule of calculus is again used, as

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{2.46}$$

Here, the sum refers to all nodes to which node j sends connections. By substituting the expression for δ in (2.43) into (2.46), the *backpropagation formula* is defined as

$$\delta_j = g'(a_j)\sum_k w_{jk}\delta_k \tag{2.47}$$

To summarize, the computation of gradients in the neural network can be explained in following steps:

1. Forward propagate the input vector $\boldsymbol{x}$ through the network by using (2.36) and (2.37) to calculate all activations z for hidden and output nodes.
2. Calculate δ for the output nodes using the observed error.
3. Calculate δ for all hidden nodes in the network using the backpropagation formula in (2.47).
4. Compute the gradients by using (2.45)

(Bishop, 2006, p.241-244)

2.5.5 Error functions

The error function for the network can be defined as

$$E(w) = \sum_{n=1}^{N} E_n(w) \tag{2.48}$$

Where $E_n(w)$ is the error function for a particular path n. There are various error functions to consider. Below, some of the most commonly used error functions are listed:

- **Cross Entropy**

$$E(w) = \sum_{n=1}^{N} E_n(w) = \sum_{n=1}^{N} \Big(y_n log(\hat{y}_n) + (1 - y_n) log(1 - \hat{y}_n) \Big) \tag{2.49}$$

  which measures the divergence between the predicted outcome $\hat{y}$ and the true outcome y. One commonly used for binary classification, since it is proven to improve training speed and reduce overfitting, compared to other models.

- **Mean squared error**

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{2.50}$$

  This is one of the simplest error functions, that basically measures the sum of the squared errors between the true value and the predicted value of the classifications. (Bishop, 2006, p.227-240)

2.5.6 Optimization algorithms

Different variations of Gradient Descent optimization are by far the most commonly used optimization algorithms when it comes to neural networks. Depending on the amount of information used to update the weights, there will be a trade-off between variance of the updates and computation time.

Batch Gradient Descent

The Batch Gradient Descent approach performs each update (or epoch) using data from the entire training set, which consequently means that it becomes increasingly slow for larger data sets since gradients must be computed for the entire data set in each update. This is done by:

$$w_{n+1} = w_n - \eta \cdot \nabla_w E(w_n) \tag{2.51}$$

Where η is the learning rate.

Stochastic Gradient Descent

Instead of using the entire training set to perform one update, Stochastic Gradient Descent uses only one sample at a time for each update.

$$w_{n+1} = w_n - \eta \cdot \nabla_w E\Big(w_n; x^{(i)}; y^{(i)} \Big) \tag{2.52}$$

This yields a much faster algorithm than Batch Gradient Descent. However, since only one sample is used at every update, it results in a higher variance between updates.

Mini Batch Gradient Descent

This method combines the speed of Stochastic Gradient Descent and the accuracy and stability of Batch Gradient Descent by creating subsets of the training data with m samples in each subset and performs each update using only one subset at a time. An update thus becomes

$$w_{n+1} = w_n - \eta \cdot \nabla_w E\Big(w_n; x^{(i:i+m)}; y^{(i:i+m)} \Big) \tag{2.53}$$

This is a popular approach used in neural networks, since it is both fast and robust.

Adaptive Moment Estimation (Adam)

A common problem in gradient descent optimization is to choose the learning rate. If the learning rate is too high, the algorithm will overshoot and having difficulties to converge to a local minimum. If the learning rate is too low on the other hand, the convergence will be extremely slow. Furthermore, the same learning rate is used for all parameter updates, which is not ideal since the input features have different characteristics. There is also a concern that the algorithm tends to get stuck in suboptimal local minima such as saddle points, where all gradients are close to zero. There are however various methods that recursively updates the learning rate that can deal with these issues. Adam is a method that computes an adaptive learning rate by storing an exponentially decaying average of past gradients m_n as well as the uncentered variance of past gradients v_n. This can be expressed as

$$m_n = \beta_1 m_{n-1} + (1 - \beta_1) g_n \tag{2.54}$$

$$v_n = \beta_2 v_{n-1} + (1 - \beta_2) g_n^2 \tag{2.55}$$

Where g_n is the gradient of the error function and β_1 and β_2 are momentum parameters that define how much weight to add from the past time step to the current update vector. Proposed values are $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

The estimates of m_n and v_n are, especially in the initial steps, subjected to a bias towards 0 since they are calculated from zero vectors. To avoid this, following modification is made:

$$\hat{m}_n = \frac{m_n}{1 - \beta_1^t} \tag{2.56}$$

$$\hat{v}_n = \frac{v_n}{1 - \beta_2^t} \tag{2.57}$$

Using (2.56) and (2.57) the update of the weights becomes

$$w_{n+1} = w_n - \frac{\eta}{\sqrt{\hat{v}_n} + \xi} \hat{m}_n \tag{2.58}$$

Where ξ is a smoothing term with default value 10^{-8} to avoid issues with dividing by zero in the case of $\hat{v}_n$ taking infinitesimal values. (Ruder, 2017)

2.5.7 Regularization

Early stopping

Early stopping can be seen as a time-based regularization. Basically, both training set error and test set error are considered. For the optimization algorithms used, the error is usually a non-increasing function of the iteration index. The test set error on the other hand generally shows a decrease initially, but at a certain point starts to increase as model complexity grows. By using early stopping, training can be stopped at the point where the test set error reaches its minimum value, which ultimately is the performance goal of a well-functioning model. (Bishop, 2006, p.259-260)

Dropout layers

With increasing complexity, the neural network is able to find complex relations in the training data, but it also increases the generalization error. This is especially the case when there is a shortage of training data, where sampling noise that is not present in the test data might corrupt the model. This may lead to poor generalization and an increasing test error. Dropout is a technique that prevents overfitting and also works as an approximation of combining exponentially many different network architectures without being computationally expensive. Dropout nodes

with all its ingoing and outgoing connections are temporarily removed by random. In Figure 2.4, the dropout procedure for a neural network with two hidden layers is shown.

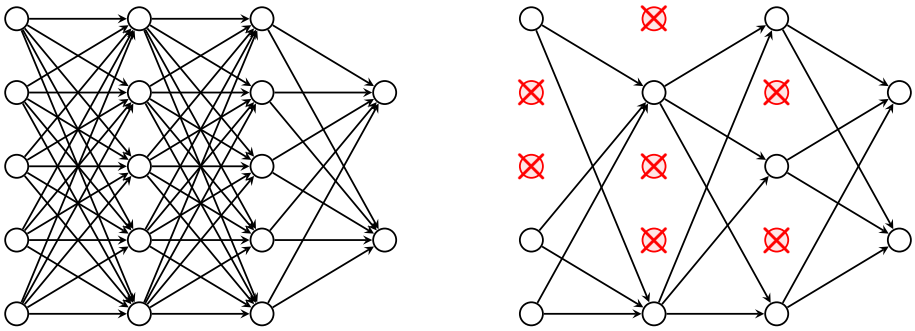

Figure 2.4: Illustration of the dropout layer procedure.

For a neural network with n nodes, there are 2^n possible thinned networks. Since they still share weights, the computation of the parameters will remain $O(n^2)$ complex. So, using dropout essentially means that instead of training a network with complexity $O(n^2)$, a set of 2^n thinned networks with extensive weight sharing is trained where each of the thinned networks are trained very rarely. When the model is used on the test data, the thinned networks are scaled down and combined into one single neural network. This has shown to significantly improve generalization errors in a variety of classification problems and outperform other regularization techniques such as early stopping. (Srivastava et al., 2014)

L1 and L2 Regularization

When using L regularization, the error function is modified with a regularization function that penalizes complexity. The modified error function becomes

$$\hat{E}(w) = E(w) + \lambda Complexity(h) \tag{2.59}$$

Where λ is the regularization term. The complexity is a function of the weights

$$Complexity(h_w) = L_q(\boldsymbol{w}) = \sum_i |w_i|^q \tag{2.60}$$

L1 regularization is the sum of absolute values of the weights, i.e. $q = 1$. For L2 regularization the function is the sum of squared weights, i.e. $q = 2$. In practice, L2 is the most favored method compared to L1 regularization when using neural networks. (Russell and Norvig, 2010, p.721)

2.6 Voting classifiers

Instead of relying on one classifier alone, one idea is to combine several classifiers to reduce variability and receive more accurate predictions. Voting models is a simple concept where several different machine learning classifiers are trained individually and combined into a voting model.

2.6.1 Hard voting

A hard voting model works by the principle of majority voting. Simply, it will classify a given data sample according to what the majority of a set of classifiers have predicted. Assume that a set of five classifiers are used in the hard voting model for a binary classification problem y $\in \{0,1\}$ with following predictions:

$$\begin{aligned} \text{Classifier 1} &\rightarrow \quad 0 \\ \text{Classifier 2} &\rightarrow \quad 1 \\ \text{Classifier 3} &\rightarrow \quad 1 \\ \text{Classifier 4} &\rightarrow \quad 0 \\ \text{Classifier 5} &\rightarrow \quad 0 \end{aligned} \tag{2.61}$$

The hard voting model will then classify the data sample as a 0 according to the majority of the voters. In case of a tie, there is a ranking order where the higher ranked classifier will have a casting vote.

2.6.2 Soft voting

The soft voting model (or soft voting ensemble) instead considers the average weighted probability computed by its classifiers, where each classifier is assigned a weight w. Assume that a set of five classifiers are used in the soft voting model for a binary classification problem. For simplicity, the classifiers are weighted equally with $w_1 = w_2 = ... = w_5 = \frac{1}{5}$. The prediction for a soft voting ensemble will be computed as the weighted average of the predictions for the classifiers:

Classifier	Weight	Output {0,1}
Classifier 1	0.2	0.1
Classifier 2	0.2	0.5
Classifier 3	0.2	0.8
Classifier 4	0.2	0.2
Classifier 5	0.2	0.4
Soft voting ensemble	1	0.4

Table 2.1: Illustration of a soft voting ensemble

Assuming that the threshold is set at 0.5, the soft voting model will classify this particular data sample as a 0. Compared with the hard voting model, the soft voting model takes into account the level of certainty of its classifiers rather than an absolute vote. This usually contributes to a higher model performance. (Pedregosa et al., 2009–2018, Section 1.11.5)

2.7 White-boxing techniques

For simple models such as a logistic regression model, the predictions are transparent and it is easy for the user to understand the classification of a specific data sample. For more complex models however, such as neural networks or support vector machines, it is not possible for a human to understand the basis of every classification. (Ribeiro et al., 2016)

In general, the human interaction in machine learning is indeed important, and the ability to understand the model is crucial for a credit analyst when making credit assessments. Primarily, it could be of interest to be able to motivate to the loan applicant why the application was rejected. Secondly, not understanding how the classifier works means that there is no way of knowing that the model actually uses relevant features to make predictions or if the predictions are made based on sample noise. If a model cannot be trusted it surely will not be used.

2.7.1 Local Interpretable Model-Agnostic Explanations (LIME)

LIME is a tool developed to easier understand a model's individual predictions by displaying the significance of the most important features for the prediction of a specific sample. The explanation of a model is defined as $g \in G$, where G is a class of interpretable models such that a human can interpret the model with the help of visual aid. To measure the complexity, the complexity of g is defined as $\Omega(g)$. In the case of a decision tree it could be the depth of the tree, and for a neural network it could be the number of nodes. Further, the model that is explained is denoted $f : \mathbb{R}^d \to \mathbb{R}$. For classification, $f(x)$ is the probability that the data sample x belongs to a certain class. Further, $\pi_x(z)$ is the proximity measure between an instance z to x, defining the locality around x. Combining these expressions, the faithfulness of the explanation is given by $L(f, g, \pi_x)$. In order to achieve an interpretable as well as locally accurate explanation, L is minimized while $\Omega(g)$ is kept at a low enough level for a human to understand. The LIME explanation is computed by

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}}\, L(f, g, \pi_x) + \Omega(g) \tag{2.62}$$

The LIME algorithm can be explained in following steps:

Require: Classifier f, Number of samples N

Require: Instance x, and its interpretable version x'

Require: Similarity kernel π_x, Length of explanation K $Z \leftarrow \{\}$

for $i \in 1, 2, 3, ..., N$ **do**
$z_i' \leftarrow$ sample around(x')
$Z \leftarrow Z \cup \left(z_i', f(z_i), \pi_x(z_i)\right)$
end for $w \leftarrow K - Lasso(Z, K)$ ▷ with z_i' as features, $f(z)$ as target
return w

Essentially the LIME algorithm bases explanations on the feature values of samples close to the sample being explained. The output is a vector containing positive and negative weights assigned to each of the used input features. By analyzing these weights, one can understand which features that the underlying model used and in which way it is important for the individual classification decision. (Ribeiro et al., 2016)

2.7.2 Shapley Additive Explanation (SHAP)

The Shapley value is a solution concept derived from cooperative game theory. Originally, it is a way to distribute the total gains to all players in a cooperation given each player's contribution. In the case of white-boxing techniques, the Shapley Additive Explanation (SHAP) value ϕ_i is found by computing the effect of including a specific input feature in the set of features used to train a model. This is done by computing a model $f_{S\cup\{i\}}$ where feature i is present, and one model f_S where the feature is excluded, and then use the difference. This can be written as

$$\phi_i = \sum_{S\subseteq F\setminus\{i\}} \frac{|S|!\big(|F|-|S|-1\big)!}{|F|!}\Big[\boldsymbol{f}_{S\cup\{i\}}(x_{S\cup\{i\}}) - \boldsymbol{f}_S(x_S)\Big] \tag{2.63}$$

Where $S \subseteq F$ is the feature subset S of F, where F is the set of all features and x_S represents the values of the input features in the subset S. SHAP values give the actual change in model performance when removing a specific feature, and is considered a more unified and comprehensive white-boxing technique than e.g. LIME. SHAP values can then be easily interpreted with visual aids in order to better understand a complex model such as a neural network better. (Lundberg and Lee, 2017)

2.8 Generalization and stability

It is reasonable to assume that characteristics of loan applicants will change slightly over time. In the case of an economic recession for example, default rates are likely to increase. Therefore, it is of high importance that the model used is stable, i.e. that minor changes in the model's input will not lead to any significant effect on the model's performance. (Bousquet and Elisseeff, 2002) Furthermore, the relatively low number of data samples available is likely to introduce variability in performance.

2.8.1 Cross-validation and selection bias

When working on small data sets with seemingly poor generalizability, one way of ensuring stability is through methods of cross-validation. The goal of the model is to accurately predict defaults in a later point in time. One could therefore trivially choose the training set as the "oldest" data points available and the test set as the "newest" to test how well the model is performing with changes in time as the characteristics of the data changes. However, with a small number of samples to begin with, problems like overfitting and selection bias are likely to occur when training and testing only is performed on one train-test split. Ideally, the model should perform well on any split to ensure that it is generalizable and stable for future use. (Cawley and Talbot, 2010)

Cross-validating is essentially splitting the data set into many train-test splits (folds), re-training, testing and measuring performance of each model on each fold and then averaging the performance. Furthermore, it is of importance that the performance does not vary too much between the splits to ensure stability. One concern is how many folds that should be created when cross-validating. There are exhaustive cross-validation methods that test performance on every possible fold. Of course, the number of possible folds varies greatly with how the size of the test (validation) set is chosen. Nevertheless, exhaustive cross-validation methods will for most practical data sets require excessive computation power. For practical purposes, a non-exhaustive cross-validation method is often sufficient. The perhaps most commonly used method is the k-fold cross-validation. The k indicates how many folds that should be created, e.g. *10*-fold indicates re-training and validating over ten randomly selected splits. For some purposes, especially when dealing with imbalanced data sets, the k-fold approach can be extended with the condition of choosing the splits randomly but stratified, thus ensuring the same class distribution in each fold. (Kohavi, 1995)

Chapter 3

Dealing with small and imbalanced data sets

Given an imbalanced data set, an appropriate way of measuring model performance has to be determined. Only considering accuracy when having an imbalanced binary classification problem can be problematic. Consider a data set of x-ray scans of potential tumors, with binary outcomes balanced 99 to 1 for "not tumor" and "tumor" respectively. Then, the naive guess of assigning all data samples to the largest category ("not tumor") will achieve an accuracy of 99%. This yields a situation where evaluating model performance becomes difficult as accuracy does not reflect the ability of the model to identify the minority class.

3.1 Relevant evaluation metrics

3.1.1 Confusion matrix and common metrics

When evaluating a binary classification method, where in this particular problem the outcome can either be classified as "bad" (default) or "good" (no default), there are four different possible events:

- True positive: The outcome is correctly predicted as "bad".
- False negative: The true outcome is "bad", and the prediction is "good".
- True negative: The outcome is correctly predicted as "good".
- False positive: The true outcome is "good", and the prediction is "bad".

These four possible events form the confusion matrix, from where many common performance metrics are derived.

		Actual value		
		p	**n**	**total**
Prediction outcome	**p′**	True positive	False positive	P′
	n′	False negative	True negative	N′
	total	P	N	

Figure 3.1: Confusion matrix, showing the distribution of correctly and incorrectly predicted outcomes

Accuracy
The total number of correctly classified outcomes in relation to the total number of samples in the test set, given by

$$\text{Accuracy} = \frac{TP + TN}{P + N} \tag{3.1}$$

Precision
The number of correctly classified "bad" outcomes in relation to the total number of samples that were classified as "bad" outcomes in the test set, given by

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.2}$$

Recall/Hit rate/True positive rate
The number of correctly classified "bad" outcomes in relation to the true number of "bad" outcomes in the test set, given by

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

False positive rate
The number of "good" outcomes incorrectly classified as "bad" in relation to the true number of "good" outcomes in the test set, given by

$$\text{False positive rate} = \frac{FP}{FP + TN} \tag{3.4}$$

F1-score
The harmonic average of precision and recall, given by

$$\text{F1-score} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{3.5}$$

3.1.2 Average precision score

For classifying outcomes of loan applications, what seems most important is the relationship between precision and recall. It is possible to model a classifier that has very high precision but captures very few of the "bad" customers. Conversely, a model configuration could find a high rate (recall) of the "bad" customers, but also incorrectly classifying a high rate of "good" customers as "bad". Ultimately it will come down to a trade-off between the loss of a defaulting customer and the alternative cost of rejecting a non-defaulting customer in terms of lost revenue. Since this relationship is not truly known, the overall performance of precision against recall needs to be taken into account on the intervals that are deemed relevant.

F1-score includes both precision and recall but is considered a quite inaccurate metric for this specific problem. Since it is the harmonic average, it doesn't capture the full relationship between the two, rather a blended metric. Also, it can only describe model performance for a specific precision/recall threshold and not the overall performance of a model on a range of thresholds.

Receiver Operating Characteristics (ROC) is a common way of comparing models used on imbalanced data sets. ROC graphs are two-dimensional graphs plotting the relationship between true positive rate (recall) and false positive rate for different thresholds. To compare performance of different models, the Area Under the ROC Curve (AUC) is computed, and the higher the AUC the better the model performance. (Fawcett, 2006)

For this specific problem an alternative to ROC is used called Average Precision score (AP), where false positive rate is replaced by precision. Average precision is computed as the weighted mean of precision for a sequence of thresholds, and weighted by the marginal increase in recall from the preceding threshold. When increasing the threshold, recall will decrease and precision will increase. The AP score is given by

$$AP = \sum_{n=1}^{N} (R_n - R_{n-1}) P_n \tag{3.6}$$

Where P_n and R_n are the corresponding precision and recall pairs at threshold n. Essentially, AP score is an approximation of the intergral of the precision-recall curve. (Buitinck et al., 2013)

Average precision can then be computed for threshold intervals that are the most relevant for the specific task of credit scoring. When having classification problems with limited amount of data samples and features with relatively weak prediction power, one can expect it to be hard to maintain a decent precision for high recall levels. Therefore, it is not as relevant to examine model performance on too high recall levels since the corresponding precision would be insufficient. Likewise, the performance on too low recall levels are not believed to be relevant either, since when implementing the classifier it would have too limited impact regardless of the corresponding precision if it can't capture enough "bad" customers.

3.2 Combating imbalance through oversampling of minority class

3.2.1 SMOTE

Synthetic Minority Oversampling Technique (SMOTE) is a method used for data sets with large imbalances between classes. It creates synthetic samples of the minority class placed on line segments joining any/all of the k nearest neighbors. The creation of one synthetic sample works in the following way:

1. Pick a randomly chosen data sample from the minority class that is being oversampled
2. Compute the Euclidean distance between the feature vector of the data sample and its k nearest neighbors
3. Multiply by a randomly generated number $\in [0, 1]$, and add it to the data sample picked in (1). Now, k new samples has been synthesized between the data sample under consideration and its k nearest neighbor.

Depending on the severity of the imbalance, one can determine how much oversampling is needed. By adjusting the number of neighbors to create synthetic points between, the amount of oversampling can be controlled.

SMOTE has also been generalized in order to handle both continuous and categorical features. This is done by also computing the median of the standard deviations considering all data samples belonging to the minority class. For each categorical variable differing from its nearest neighbors, the computed median of standard deviations is added to the Euclidean distance. The categorical feature is then picked based on the majority of its k nearest neighbors. (Chawla et al., 2002)

3.2.2 ADASYN

Adaptive Synthetic Sampling (ADASYN) uses the same basic idea of SMOTE by constructing artificial samples using the Euclidean distance between a data sample and its k nearest neighbors. The difference is that ADASYN uses a weighted distribution to generate synthetic data based on information from minority class samples that are the most difficult to learn. The method can be described in following steps:

1. For each minority class data sample, calculate the ratio of the k nearest neighbors belonging to the majority class samples:
$$r_i = \frac{\nabla_i}{k} \tag{3.7}$$
where ∇_i is the number of majority class samples among the k nearest neighbors.
2. Normalize r_i so it becomes a probability distribution:
$$\hat{r}_i = \frac{r_i}{\sum_i r_i} \tag{3.8}$$
3. Calculate the required number of synthetic samples created for each minority sample.
$$g_i = \hat{r}_i \cdot G \tag{3.9}$$
where G is the total number of synthetic data samples created, depending on the imbalance in the data and the desired outcome.
4. Repeat for all minority samples in the data set and create synthetic samples in the same way as SMOTE by calculating the Euclidean distance.

By weighting minority class samples with many majority class neighbors (i.e. outliers) heavier when synthesizing new samples, the classifier models will easier identify and learn these hard-to-classify cases. (He et al., 2008)

3.2.3 Weighting

A simple yet effective way of dealing with imbalanced classes is to configure the model's loss function by penalizing misclassification of the minority class. This is done by assigning a larger weight to minority class samples in the loss function in order to put more emphasis on accurately predicting this class. Essentially, it has the same effect as copying and adding up minority class samples in the training set to a desired level. By using trial and error, this method can improve model performance to a certain degree when handling imbalanced data sets. (Pedregosa et al., 2009–2018)

3.2.4 Undersampling

There are several techniques referred to as undersampling, which means that the class distribution is balanced by removing samples from the majority class to achieve a desired percentage of minority samples in the training set. The easiest way is by simply removing random samples from the majority class until the desired ratio is reached. It is an effective and simple approach when having vast amounts of data. (Chawla et al., 2002) However, considering the amount of data available, the idea of undersampling was not considered since the actual problem lies within the lack of minority class data rather than the imbalance itself. Undersampling an already small data set would likely not improve performance.

Chapter 4
Implementation

4.1 Data set

4.1.1 Removing recurring customers

Initially, an attempt to classify both new and recurring customers with the same algorithms was made. However, new and recurring customers were proved to be considerably different, with recurring customers' risk being explained by variables that are not available for new customers. Since new customers are more difficult for the Company to predict and that they tend to default more often, recurring customers were excluded from the data set. This narrows it down from 8771 samples to 4052 unique data points. Out of these, 17.8% are labeled as defaults.

4.1.2 Pre-Processing

An initial cleaning of the data set was made, where input features with no discriminatory power such as ID-number, dates and other irrelevant information was removed. Furthermore, input features that had very few entries were removed. An outlier detection was also performed to catch wrongly generated data caused by system errors that could harm the performance of the classifiers. Most of the data was however complete and there were no issues with replacing missed values. Dummy variables denoted 0 and 1 were also constructed for selected features, in order to be able to use as input during training of the models.

4.1.3 Training and test splits

The data set was split into a training set and a test set, with a ratio of 80/20. This means that the training set consists of 3242 samples (80%) and the test set 810 samples (20%). The data was shuffled to ensure that no time dependency or seasonality could corrupt the split. The shuffling was made with a fixed random seed in order to evaluate the model performance during training. Moreover, the splits were stratified in order to maintain the same ratio between "good" and "bad" customers at 17.8%.

4.2 Feature Engineering

Recognizing both the shortage of data as well as an uneven class distribution between "good" and "bad" outcomes, there are several things to take into account. Not having a large enough training set restricts the number of input features that can be used without causing overfitting, commonly referred to as "curse of dimensionality". Therefore, it was decided to carefully investigate all available input features to attain the most suitable set of features.

In the data, there are several input features to take into account that were retrieved in the initial loan applications. They were grouped into:

- **Numerical features**
  This includes continuous variables such as monthly income, monthly expenses, age and what amount of credit that was applied for.
- **Binary features**
  This includes features with two outcomes, such as gender, marital status, property owner (y/n) etc.
- **Qualitative features**
  This includes features with several outcomes and no internal ranking, such as loan purpose, family constellation and geographic location of residence.

All available input features before any re-coding are listed in Table A.1 in Appendix A.

4.2.1 Finding significant input features

Categorization

When making efficient categorization of both numerical and qualitative features, there are three important factors to take into account:

1. Each category should consist of a minimum number of "good" and "bad" loans. Creating too small categories with too few "bad" samples may hurt both accuracy and generalizability.
2. The samples in each category should have similar risk profiles. When creating categories for different income levels for example, it is important to find income groups that show similar risk patterns (i.e. low-income, middle-income, high-income individuals).
3. The resulting categorization should have a plausible risk pattern (given by Weight of Evidence) and a high performance (given by Information Value).

To evaluate the third criteria, Weight of Evidence (WoE) and Information Value (IV) are two metrics being used. These metrics are recommended when choosing and categorizing input features used in scoring models especially for retail exposures, where qualitative variables such as marital status, loan purpose etc. is predominant. WoE is a metric that expresses the discriminatory power of a category within an input feature, and is calculated from the log percentage relation of "good" vs "bad" outcomes within that category, i.e.

$$WoE_c = ln\Big(P(c|good)) - ln(P(c|bad)\Big) \tag{4.1}$$

Where WoE_c is Weight of Evidence for category c of a given input feature. From WoE, the IV of an input feature with a given set of categories C are calculated from the percentage relation weighted by WoE for each c, i.e.

$$IV_C = \sum_{c=1}^{C} WoE_c\Big(P(c|good) - P(c|bad)\Big) \tag{4.2}$$

Making the right categorization of input features is a complex process, since there is a trade-off between the requirements mentioned before. Having many and small categories will increase the IV but might affect the final performance of the model negatively in terms of poor generalizability. On the other hand, having too few categories will make the model too general and unable to make accurate predictions. (Engelmann and Rauhmeier, 2011, p.34-35)

Manual selection

Since credit assessment and predicting credit defaults is an area that has been deeply explored historically, there are several characteristics widely known that intuitively should indicate a higher risk such as an applicant's income or the size of the loan. Given that many of the available inputs are conceptually relevant combined with the results from the Information Value tests, the list of input features to examine further was narrowed down.

Initially, each input feature was visualized separately against the default frequency. In Figure 4.1 it is illustrated that if the applicant owns a property, the average default frequency is significantly lower than for customers that do not own a property. It is also shown that property owners constitute 14% of the data set, which is considered a large enough category. This initial approach was applied repeatedly to remove features that shown no difference in terms of discriminatory power, and to remove or in some cases merge categories that were too small (<5% of the total population). Using this procedure, the remaining input features were then ranked based on discriminatory power and tested one by one to see the impact of adding and removing one input feature at a time to achieve an optimal set of features.

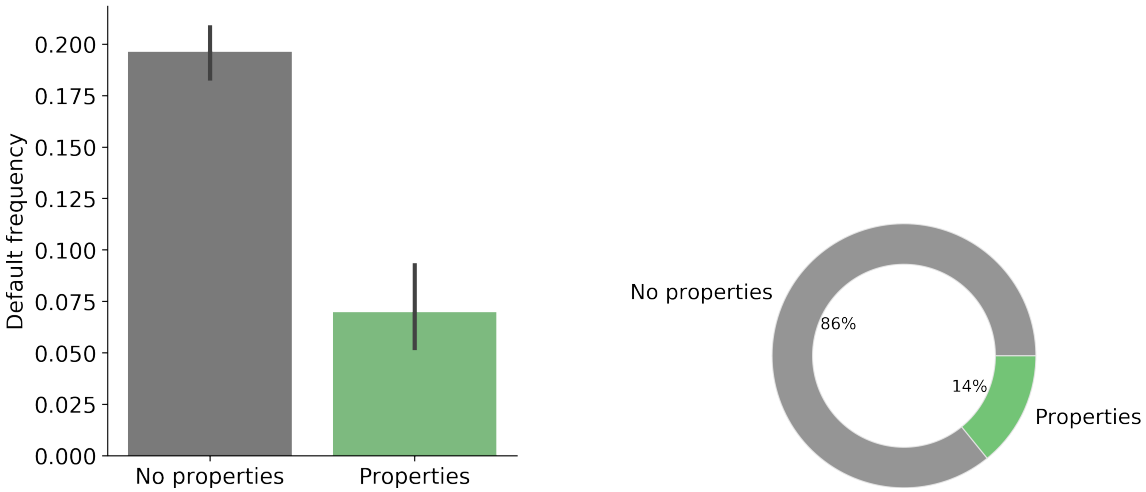

Figure 4.1: Default frequency (left) and distribution (right) of customers with and without a property

Permutation importance

To further ensure that the selected features contribute to the desired performance metric the method of permutation importance was used. It is a crude method consisting simply of scrambling the values of one input feature several times and measuring the difference in the desired performance metric. In this instance, average precision in the interval 20-50% recall was used as the desired metric. The values of each input feature vector were scrambled separately and the loss in performance was measured one by one and ranked accordingly. (Breiman, 2001, p.18-20)

Dealing with input correlation

In linear and logistic regression, multicollinearity is an issue, meaning that it hurts a model's stability to use input features with very high correlation. On the other hand, non-parametric models such as support vector machines and neural networks are not as easily affected by high input correlation. Therefore, the models that were theoretically affected by multicollinearity were only trained on independent input features. The highest observed correlation between used inputs was 0.52 which should not be an issue in terms of multicollinearity. (Bolton, 2009, p.27) In Figure 4.2, a correlation heatmap for selected variables is shown.

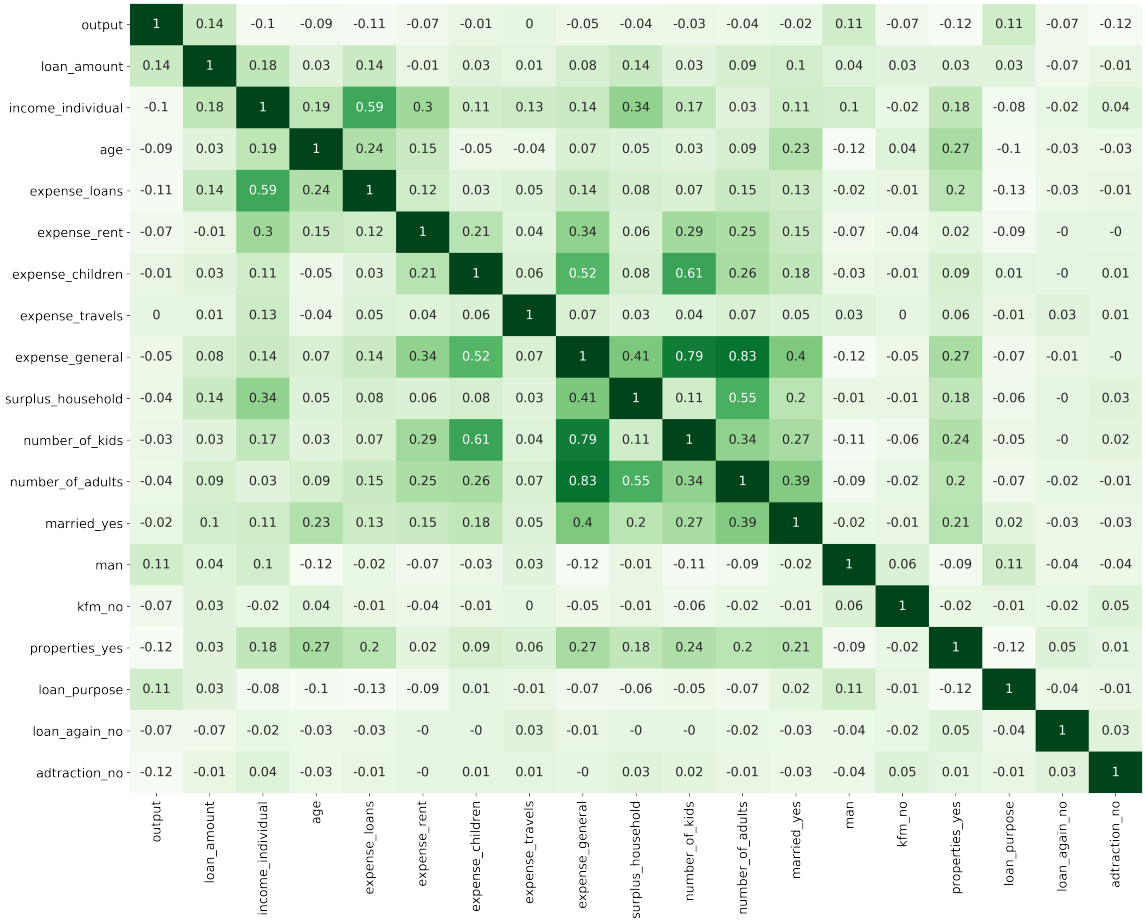

Figure 4.2: Correlation heatmap showing high positive correlation in dark coloring, and no or negative correlation in light coloring.

4.2.2 Construction of optimal feature configurations

Using the aforementioned methods and evaluation criteria, the input features were carefully refined and grouped accordingly to ensure high discriminatory power as well as generalizability, and is shown in Appendix A. In Table A.2, Feature Subset 1 is shown, where all input features were re-coded as dummy variables, i.e. only taking values 0 or 1. In Table A.3, Feature Subset 2 is shown, where numerical variables such as income and age were categorized into 3-5 categories, combined with dummy variables such as gender and property ownership. In Table A.4, Feature Subset 3 is shown were numerical features remained untouched. A combination of these subsets was constructed to generate the best performance results and is shown in Table A.5, later referred to as Feature Subset 4.

4.2.3 Separating test sets by loan amount

Large loans generate more revenues and are - when paid back in full - more profitable, since marketing and handling costs are dependent on the number of loans rather than capital lent, leading to lower marginal costs the higher the loan amount. They are however more risky, both in terms of amount of capital at risk of default and costly in terms of tied up capital.

Loan amount seems to affect risk of default considerably. Larger loans tend to be riskier where 25% of every loan above 20 kSEK defaults while less than 10% of small loans (<4 kSEK) defaults. Of the total number of approved loan applications, large loans (>20 kSEK) account for 26%. Of the total principal amount of the lending volume however, large loans account for 53%, and of the total principal amount of loans that defaulted, large loans account for 62%. Hence, it is critical for the models to perform well on large loans, in terms of both precision and recall to prevent credit losses without rejecting too many "good" customers. Therefore, the models were evaluated on how well they performed depending on the loan amount for each loan application, in order to ensure that large loans were at least as accurately predicted as the smaller loans. Three subsets of the test set were constructed for the intervals 1-4kSEK (small loans), 5-20 kSEK (mid-sized loans) and 21-25 kSEK (large sized loans) to be tested separately in Section 5.2.

The default frequencies and percentage of total number of loans in the data set for the three subsets are illustrated in Figure 4.3. Remember that the subset of large-sized loans only constitutes 26% of the number of loan applications, but 53% of the total lending volume in terms of principal amount.

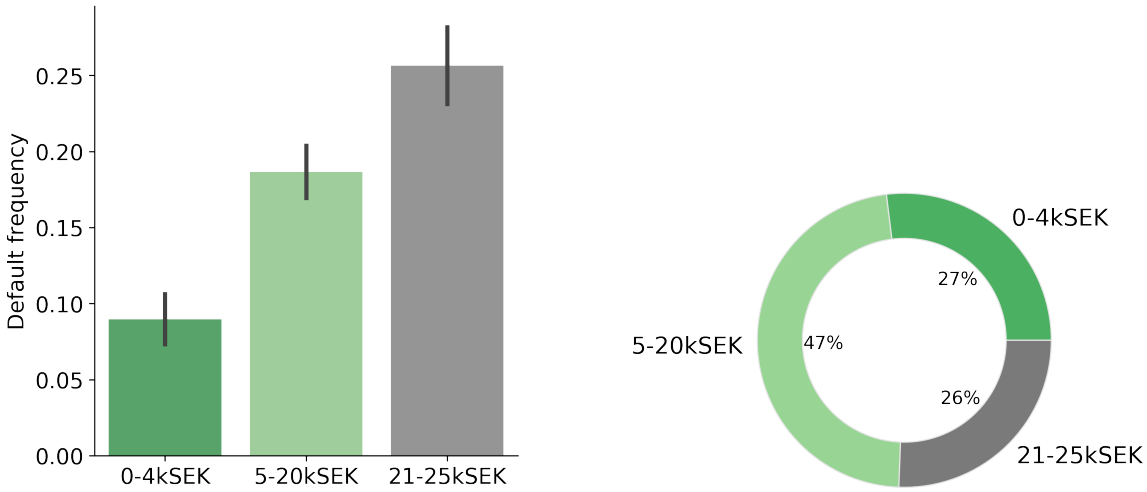

Figure 4.3: Default frequency (left) and distribution (right) for customers in the three loan amount categories

4.3 Classifier configuration and tuning

Initially, Feature Subset 1, 2 and 3 shown in Appendix A were tested on the models to investigate whether one model would achieve higher performance than another on a different feature subset. Finally a combination of the three subsets was chosen, referred to as Feature Subset 4, and is listed in Table A.5 in Appendix A. Feature Subset 4 was chosen since it lead to better performance in all instances, as well as simplifying the benchmarking process between the models by using the same inputs.

The models were implemented using *scikit-learn*, an open source Machine Learning Package in Python that provides efficient implementations of various Machine Learning algorithms and metrics tools. The model hyperparameters were then tuned by using a modified implementation of average precision, allowing for computing average precision over specified recall intervals rather than 0 to 100%. Grid search was used to tune the models. Grid search is a brute-force approach which basically means that a large amount of model parameter combinations is evaluated using cross-validation, and then the combination with the best performance (in terms of average precision in this case) is chosen. When a decent model parameter combination was obtained, manual fine-tuning was performed to push performance further.

The model output is a value $0 \leq \hat{y} \leq 1$. In order to classify a sample, a threshold u is defined, where $0 \leq u \leq 1$. If the output value $\hat{y}$ is higher than u, the data sample is classified as "bad", and classified as "good" if $\hat{y}$ is lower than u. This also allows for risk adjusting without re-training the models, which is convenient when used in practice. By increasing the threshold u, precision increases and recall decreases, and vice versa when lowering u.

4.3.1 Logistic regression

The model was trained using the Newton's conjugate gradient algorithm. Class-weights were set to 1:1 since no improvement was shown by altering the weights.

4.3.2 Decision tree

The minimum sample split and minimum samples per leaf were set to 2 and 1 respectively. The optimal max depth d was determined using grid search over the interval $d = 1, 2, \ldots, 19, 20$. Final max depth was set to $d = 8$. The quality of the splits was evaluated using Gini index. Class-weights were set to 1:1 since no improvement was shown by altering the weights.

4.3.3 Random forest

A grid search was performed to determine the number of trees/estimators m over the interval $m = 1, 2, \ldots, 199, 200$ and the tree depth d over the interval $d = 1, 2, \ldots, 19, 20$. The quality of the splits was evaluated using Gini index, and the number of features considered for each split was set as the square root of the number of trees $\sqrt{m}$, which is the recommended setting. Final number of trees was set to $m = 100$ and depth of tree set to $d = 7$. Class-weights were set to 1:1 since no improvement was shown by altering the weights.

4.3.4 k-Nearest Neighbors

The number of neighbors k to include in the algorithm was set using grid search between $k = 1, \ldots, 99, 100$. The final number of nearest neighbors was set to $k = 30$. Class-weights were set to 1:1 since no improvement was shown by altering the weights.

4.3.5 Support vector machine

Initially, the kernel type was set to *'linear'*, which is the linear kernel function. The reason was that the non-linear kernels *'poly'*, *'rbf'* and *'sigmoid'* required unmanageable computation time and did not give any satisfying results despite various attempts.

When using the linear kernel function, the cost function C and class weights were the only hyperparameters needed to be tuned. They were determined using grid search for the interval $C = 10^{-5}, 10^{-4}, \ldots, 10^{4}, 10^{5}$ and class-weights in the interval 1:1, 1:1.5,...,1:4.5,1:5. C was finally set to $C = 10$ and class-weights set to 1:4.

4.3.6 Artificial neural network

Number of hidden layers was set to one. Two or more hidden layers were tested without increase in performance and were therefore discarded. Learning rate was set to *'invscaling'* which gradually decreases the learning rate at each time step using an inverse scaling exponent. Various regularization methods were evaluated, where L2 regularization performed the best. An extensive grid search was then performed with following parameters over respective intervals:

- Number of nodes in hidden layer: $h = 20, 30, \ldots, 190, 200$.
- L2 penalty parameter: $\alpha = 0.1, 0.2, \ldots, 0.9, 0.95, 0.99, 0.999$
- Optimizer: 'quasi-newton', 'sgd', 'adam'
- Class weights: 1:1, 1:1.5,...,1:2.5,1:3

Final parameters were set to $h = 100$, $\alpha = 0.5$, class weights 1:1.5 and the *Adam* algorithm used as optimizer.

4.3.7 Voting ensemble

The soft voting model was used instead of the hard voting model since it performed better in all instances. Then, various model combinations, feature subset combinations and weightings were tried. The final version is shown in Table 4.1, where two logistic regression models, one using Feature Subset 1 in Table A.2 and one using Feature Subset 4 in Table A.5 were weighted together with a neural network trained as explained in Section 4.3.6.

Classifier	**Weight**
Logistic regression (Feature Subset 1)	0.25
Logistic regression (Feature Subset 4)	0.5
Neural network (see Section 4.3.6)	0.25

Table 4.1: Included models in the soft voting ensemble with corresponding weights

Many attempts were made to include other models in the voting ensemble, but they failed to improve performance.

Chapter 5
Results

Clarification of abbreviations and performance metrics

In this chapter, performance of five chosen classifiers is presented; 1. Logistic regression, 2. Random forest, 3. Support vector machine, 4. Neural network and 5. Soft voting ensemble consisting of two logistic regression models and a neural network. They are abbreviated in forthcoming tables as 1. LOG, 2. RFO, 3. SVC (Support Vector Classifier), 4. MLP and 5. SVO respectively. k-Nearest Neighbors and the decision tree classifier were tested but their results are left out of this chapter due to overall poor performance. In Table 5.1, the model abbreviations are listed.

Classifier	Abbreviation
Logistic Regression	LOG
Random Forest	RFO
Support Vector Machine	SVC
Neural Network	MLP
Soft Voting Ensemble	SVO

Table 5.1: Model abbreviations used in the forthcoming results tables

As one of the most important aspects of the model is flexibility to handle changes in the Company's risk appetite over time, several measures of performance are needed to evaluate the models. Precision is measured at 35% recall, this metric is called *PRX* in the following results. Average precision score is also measured at recall intervals 20-35% and 35-50% to measure performance on higher and lower recall levels. Average precision also serves as a measure of overall performance rather than measuring precision at a certain precision/recall threshold. This factor was proved to be important since precision demonstrated rather unstable behavior depending on which threshold it was evaluated at. The metrics used throughout this chapter are referred to as:

- **AP1 (Average precision at recall level 20-35%)**
- **AP2 (Average precision at recall level 35-50%)**

Moreover, the precision-recall curve is initially displayed to illustrate the precision-recall trade-off for each model where the area under the curve can be interpreted as the average precision score.

5.1 Overall performance

In this section, model performance using the full data set was performed and evaluated. In Figure 5.1 the precision-recall curve is shown for the five models under consideration.

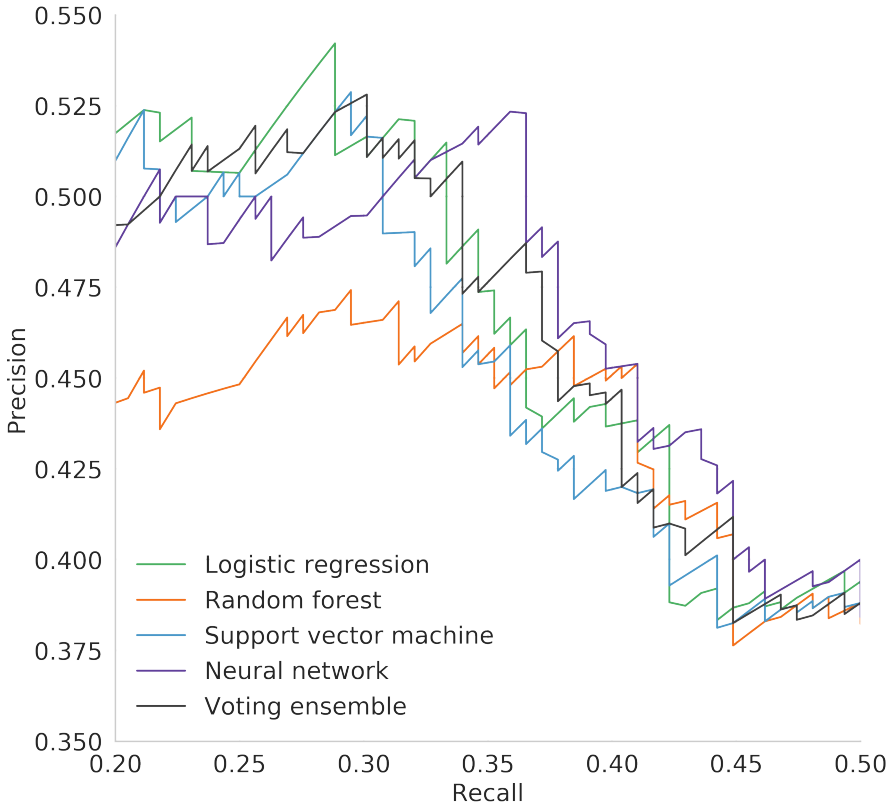

Figure 5.1: Average precision curve

As seen in Figure 5.1, most models offer similar performance and a somewhat smooth trade-off between precision and recall. Notably, the random forest classifier has a more volatile trade-off between precision and recall, as well as significantly under-performing on high precision/low recall-pairs. The soft voting ensemble's AP curve seems somewhat smoother than for the other individual classifiers. In Table 5.2, the average precision scores for recall levels 20-35% and 35-50% as well as precision at recall 35% is shown.

	LOG	RFO	SVC	MLP	SVO
AP1	0.509	0.4688	0.4956	0.4974	0.5032
AP2	0.4275	0.4371	0.4191	0.4438	0.4355
PRX	0.4825	0.4867	0.4783	0.4867	0.4786

Table 5.2: Overall performance on test set

Table 5.2 confirms what was seen in Figure 5.1; most models show similar performance with the random forest classifier as a slight under-performer on lower recall levels.

5.2 Performance on applications of varying loan amounts

For the results below, the data set was divided into three subsets as explained in Section 4.2.3:

- Subset 1: Loans between 21-25kSEK, 1037 samples with 266 defaults (26%)
- Subset 2: Loans between 5-20kSEK, 1923 samples with 359 defaults (19%)
- Subset 3: Loans between 1-4kSEK, 1092 samples with 98 defaults (9%)

The models were then trained on the original 80 % training set and tested on each and one of the three loan amount categories. The results are presented below.

	LOG	RFO	SVC	MLP	SVO
AP1	0.5632	0.5058	0.557	0.5356	0.5794
AP2	0.5055	0.4592	0.4867	0.4917	0.5061
PRX	0.5333	0.4528	0.4898	0.5208	0.5217

Table 5.3: Performance on test subset 1 (large loan amounts)

As seen in Table 5.3, models are able to reject a large portion of defaulting large loans with a precision of approximately 50%, thus indicating that the models perform better than average on large loan amounts.

	LOG	RFO	SVC	MLP	SVO
AP1	0.503	0.4954	0.4906	0.5106	0.492
AP2	0.3725	0.3741	0.3841	0.3951	0.3747
PRX	0.4528	0.5106	0.4615	0.48	0.4286

Table 5.4: Performance on test subset 2 (mid-sized loan amounts)

As seen in Table 5.4, the models are not predicting mid-sized loans as accurate as large sized loans.

	LOG	RFO	SVC	MLP	SVO
AP1	0.2038	0.2011	0.1882	0.2073	0.1904
AP2	0.1714	0.1181	0.1556	0.1744	0.1686
PRX	0.18	0.125	0.15	0.1731	0.1765

Table 5.5: Performance on test subset 3 (small loan amounts)

As seen in Table 5.5, predicting defaults of small loan amounts (>4 kSEK) is very difficult. Only 9% of these loans default which could be part of the reason. Using a fixed risk threshold over all loan amounts results in models predicting almost no defaults among small loan amounts and few among the mid-sized loans. Going forward, the reader should keep in mind that performance on large loan amounts is crucial and performance on small loan amounts is of lesser importance. During the rest of the Results chapter which will explore stability of performance, following three tables will be presented for each section, including the previously used metrics *AP1*, *AP2* and *PRX*:

1. Overall performance on the test set.
2. Performance on test subset 1 (large loan amounts).
3. Performance on test subset 2 (mid-sized loan amounts)

5.3 Performance stability testing

5.3.1 Cross-validation of performance

Since precision seems to vary a lot over different recall levels for all models (see Figure 5.1) and the test set is relatively small (only 810 samples of which 150 are denoted "bad") it is important to evaluate performance across many different train/test splits to reduce some of the variance of the performance measurements. Below, performance averaged from 18 randomized train-test splits (18-fold cross-validation) is shown. The splits were made with 70% train data and 30% test data to further help stabilize test results at expense of some performance.

	LOG	RFO	SVC	MLP	SVO
AP1	0.4446	0.4178	0.442	0.4436	0.4439
AP2	0.3789	0.3777	0.3819	0.3813	0.3809
PRX	0.4137	0.4053	0.4088	0.4151	0.4142

Table 5.6: Average performance over 18-fold cross-validation on full test set

	LOG	RFO	SVC	MLP	SVO
AP1	0.5211	0.4419	0.4552	0.5043	0.5187
AP2	0.4772	0.4218	0.4349	0.4689	0.4786
PRX	0.4989	0.43	0.4403	0.4864	0.4983

Table 5.7: Average performance over 18-fold cross-validation on test subset 1 (large loans)

	LOG	RFO	SVC	MLP	SVO
AP1	0.3803	0.3541	0.379	0.3689	0.3766
AP2	0.3434	0.327	0.3425	0.3393	0.346
PRX	0.3556	0.3415	0.3584	0.3536	0.3605

Table 5.8: Average performance over 18-fold cross-validation on test subset 2 (mid-sized loans)

After cross-validating the performance results it becomes even more clear that the performance scores are very similar for all models except random forest which shows poor performance. The support vector classifier is also performing worse than logistic regression, neural network and soft voting ensemble, especially on large loans. When estimating the standard deviations of the performance scores over the 18 different splits, about the same standard deviations were obtained for all models. For the results in Table 5.6, the standard deviations were approximately 3.5% for all models and all performance metrics. Naturally, since the subsets of large and mid-sized loans are smaller than the full set, the standard deviations of those scores were somewhat higher but again about the same across all models.

5.3.2 Data volume effect on performance

Starting out with a relatively small data set is often a challenge from a modelling standpoint. Since the relevant data is generated by new customers, it is trivially impossible to gather more data at a given point in time. This is particularly the case when limiting the data to the customers of this particular risk segment. Data generated from competitors' customers could potentially be useful if the customer characteristics was sufficiently similar. Nonetheless, disregarding the possibility of gathering more data at this point in time it is relevant to investigate how sensitive the models are to changes in training data volume. This could potentially offer some insight to how the model performance is expected to increase as time progresses and the training data naturally grows. Cross-validated performance tests were run, but training on only 7% of the available data as opposed to 70% in Section 5.3.1. The average precision score averaged over 18 train-test splits with 7% train data and 93% test data are shown below.

	LOG	RFO	SVC	MLP	SVO
AP1	0.3862	0.3376	0.3664	0.3828	0.3905
AP2	0.3409	0.3019	0.3292	0.3345	0.3443
PRX	0.3642	0.3185	0.3482	0.3577	0.3687

Table 5.9: Average performance over 18-fold cross-validation by models trained on only 7% of data

	LOG	RFO	SVC	MLP	SVO
AP1	0.4121	0.383	0.3675	0.4051	0.4114
AP2	0.3861	0.3541	0.3533	0.3747	0.385
PRX	0.3965	0.3643	0.3625	0.3855	0.3959

Table 5.10: Average performance over 18-fold cross-validation on test subset 3 (large-sized loans), by models trained on 7% of data

	LOG	RFO	SVC	MLP	SVO
AP1	0.332	0.2708	0.3115	0.312	0.3302
AP2	0.3049	0.2601	0.2854	0.2908	0.3017
PRX	0.322	0.2662	0.3008	0.3042	0.3186

Table 5.11: Average performance over 18-fold cross-validation on test subset 2 (mid-sized loans), by models trained on 7% of data

The performance drop from training on only 7% of the data is clearly shown in the above tables comparing with the results in Section 5.3.1 where 70% of the available data was used. However, the drop is not as severe as might be expected from reducing the number of training samples from 2836 samples to only 284.

To be able to further draw conclusions from the effect on performance by increasing training data volume, similar tests as above were run but with 35% of the available data used for training. The results are shown in Figure 5.12, 5.13 and 5.14.

	LOG	RFO	SVC	MLP	SVO
AP1	0.4296	0.3977	0.4257	0.4273	0.4318
AP2	0.3721	0.3567	0.3719	0.3693	0.375
PRX	0.4021	0.3783	0.3988	0.4019	0.4052

Table 5.12: Average performance over 18-fold cross-validation by models trained on 35% of data (1814 samples)

	LOG	RFO	SVC	MLP	SVO
AP1	0.4855	0.4132	0.439	0.4717	0.486
AP2	0.4512	0.3988	0.4155	0.4352	0.4505
PRX	0.4755	0.408	0.4249	0.45	0.4677

Table 5.13: Average performance over 18-fold cross-validation on test subset 3 (large-sized loans), by models trained on 35% of data

	LOG	RFO	SVC	MLP	SVO
AP1	0.3703	0.3223	0.3624	0.3634	0.373
AP2	0.3286	0.2985	0.3235	0.3216	0.3322
PRX	0.3475	0.309	0.3361	0.3402	0.351

Table 5.14: Average performance over 18-fold cross-validation on test subset 2 (mid-sized loans), by models trained on 35% of data

Comparing the results in Table 5.6, 5.7 and 5.8 (70% training data) to Table 5.9, 5.10 and 5.11 (7% training data) as well as to Table 5.12, 5.13 and 5.14 above (35% training data) it is clear that an increase in data volume has a positive effect on performance. It is however difficult to determine when or if the gain in performance from increased data volume would stagnate further on. There is no guarantee and rather unlikely that the performance gain would continue to increase with a log-linear relationship as shown in the results. This is further discussed in Section 6.3.

5.3.3 Stability of performance in time

Since the models in reality will be trained on historical loan applications and used to predict new ones, it is important that performance does not sharply drop when training on old data and testing on new. To test this, the models were trained on the "oldest" 80% of the data set and tested on the "newest" 20%.

	LOG	RFO	SVC	MLP	SVO
AP1	0.4388	0.3726	0.4338	0.4423	0.4403
AP2	0.3808	0.3648	0.3798	0.3777	0.377
PRX	0.4185	0.3775	0.4162	0.4278	0.4451

Table 5.15: Performance on test set chosen as the "newest" 20% of samples

	LOG	RFO	SVC	MLP	SVO
AP1	0.5407	0.509	0.5315	0.5182	0.5374
AP2	0.5052	0.4429	0.506	0.4925	0.4968
PRX	0.5091	0.4746	0.5185	0.5185	0.4828

Table 5.16: Performance on test subset 1 (large loan amounts) chosen as the "newest 20% of samples

	LOG	RFO	SVC	MLP	SVO
AP1	0.3878	0.3215	0.381	0.3841	0.396
AP2	0.336	0.3238	0.3329	0.3231	0.3296
PRX	0.3776	0.319	0.3663	0.3737	0.3725

Table 5.17: Performance on test subset 2 (mid-sized loan amounts) chosen as the "newest 20% of samples

Compared with the results from Section 5.3.1, the performance is in line with the 18-fold average performance.

5.3.4 Oversampling effect on performance

SMOTE and ADASYN

Attempts were made to create synthetic "bad" samples by using the oversampling techniques SMOTE and ADASYN described in Section 3.2 to generate 25% additional "bad" samples. These attempts were however unsuccessful, since they resulted in lower performance for all models. Therefore, the results are not presented nor further discussed. The believed reason for this is that the imbalance itself might not be the issue, rather the limited data volume as a whole.

5.4 White-boxing and evaluation of feature importance

5.4.1 Permutation importance

Permutation importance was evaluated with respect to the effect of each input feature on the average precision between 20-50% recall. In other words, a weight of 0.1 should be interpreted as contributing to an increase of 10% in average precision on the relevant recall levels. The permutation tests were run on both training and test data on the original split of 80% train and 20% test data. The results are shown for logistic regression and the neural network, with the results for the other models shown in Appendix B.

Weight	Feature
0.1329 ± 0.0506	loan_amount_3
0.0429 ± 0.0195	adtraction_no
0.0289 ± 0.0441	man
0.0276 ± 0.0211	properties_yes
0.0272 ± 0.0292	loan_purpose
0.0215 ± 0.0305	loan_again_no
0.0171 ± 0.0422	income_5
0.0171 ± 0.0247	expense_loans_3
0.0155 ± 0.0245	age_3
0.0100 ± 0.0456	kfm_no

Weight	Feature
0.0988 ± 0.0221	loan_amount_3
0.0441 ± 0.0167	adtraction_no
0.0332 ± 0.0138	income_5
0.0319 ± 0.0149	man
0.0311 ± 0.0136	expense_loans_3
0.0181 ± 0.0071	age_3
0.0173 ± 0.0158	properties_yes
0.0168 ± 0.0097	loan_purpose
0.0120 ± 0.0105	loan_again_no
0.0044 ± 0.0120	kfm_no

Figure 5.2: Permutation importance for logistic regression on test set (left) and train set (right)

Weight	Feature
0.1245 ± 0.0490	loan_amount_3
0.0417 ± 0.0187	adtraction_no
0.0262 ± 0.0283	loan_purpose
0.0248 ± 0.0404	man
0.0228 ± 0.0391	income_5
0.0200 ± 0.0215	properties_yes
0.0176 ± 0.0229	expense_loans_3
0.0156 ± 0.0228	age_3
0.0136 ± 0.0222	loan_again_no
0.0129 ± 0.0367	kfm_no

Weight	Feature
0.1116 ± 0.0213	loan_amount_3
0.0450 ± 0.0172	adtraction_no
0.0408 ± 0.0182	income_5
0.0382 ± 0.0185	expense_loans_3
0.0372 ± 0.0124	man
0.0235 ± 0.0085	loan_purpose
0.0217 ± 0.0144	properties_yes
0.0205 ± 0.0079	age_3
0.0121 ± 0.0112	loan_again_no
0.0093 ± 0.0104	kfm_no

Figure 5.3: Permutation importance for neural network on test set (left) and train set (right)

As seen in the above tables, the neural network seems to base predictions on very similar input features to logistic regression with loan amount being the strongest predictor.

5.4.2 Explaining overall predictions using SHAP

In Figure 5.4 and 5.5, the overall explanations for all predictions by the logistic regression model and the neural network are shown using the SHAP approach. Similar plots for the other models are found in Appendix B. A higher SHAP value indicates a higher predicted probability of default. The coloring indicates the value of the specific input feature, where red is high relative to other samples and blue represents low relative values. So, for the binary feature "man" taking values 0 or 1, a red value would indicate a high value, 1 (i.e. a male customer) and a red value (5) of "income_5" indicates the highest income category (>27.5 kSEK). The distribution of samples with respect to SHAP values is represented by the thickness of the blobs.

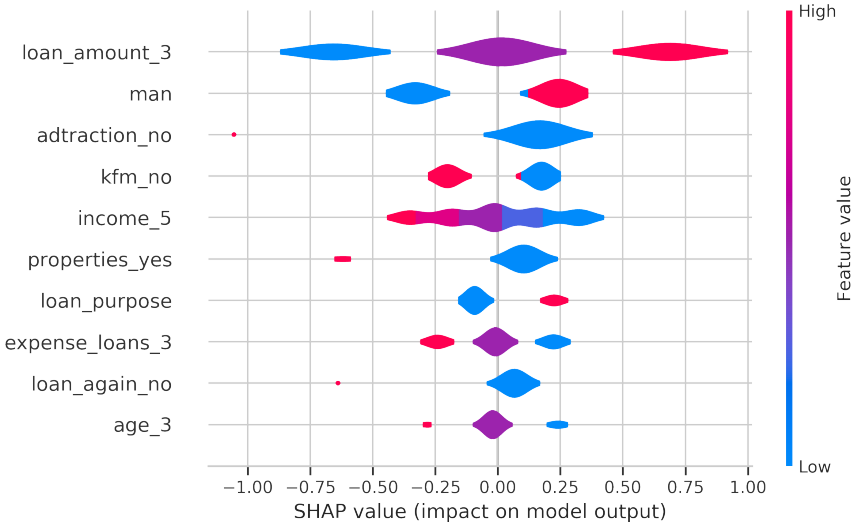

Figure 5.4: Effect on predictions by logistic regression per feature by SHAP value

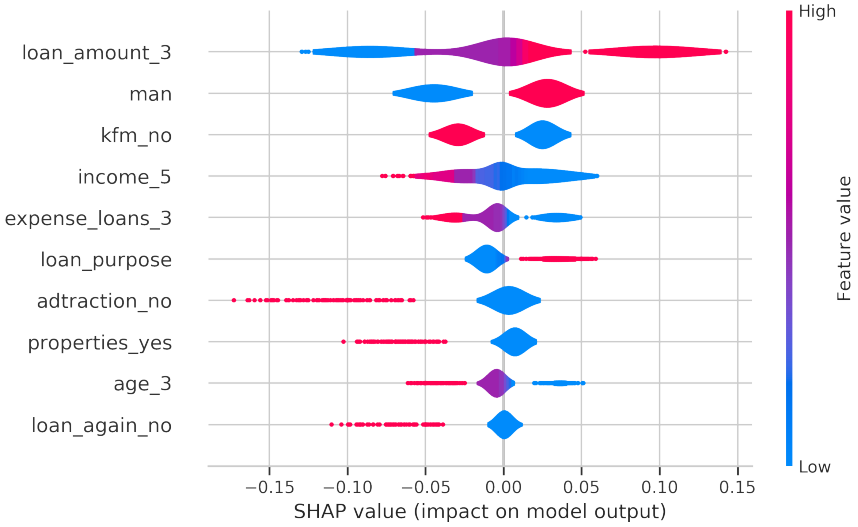

Figure 5.5: Effect on predictions by neural network per feature by SHAP value

As seen in Figure 5.4 and 5.5, the logistic regression model and the neural network bases predictions on very similar features. However, worth to notice is that the neural network does not assign the same SHAP value to each sample for e.g. $adtraction\_no = 1$.

5.4.3 Explaining individual predictions through SHAP

Below, explanations of individual predictions are made using SHAP value on predictions performed by logistic regression and neural network. Figure 5.6 and 5.7 show the explanation of a correctly classified "bad" customer for each model respectively, and Figure 5.8 and 5.9 show the explanations of a correctly classified "good" customer. The length of each segment in the diagram represents the effect on the prediction by each input feature value adding up - from the base value - to the model output. Similar plots for the other models are included in Appendix B.

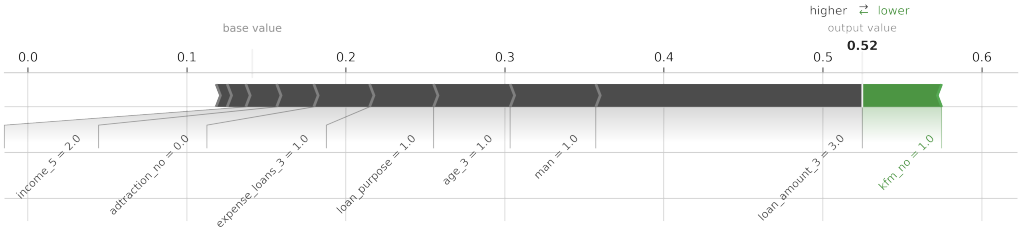

Figure 5.6: Individual explanation of LOG on a "bad" customer prediction

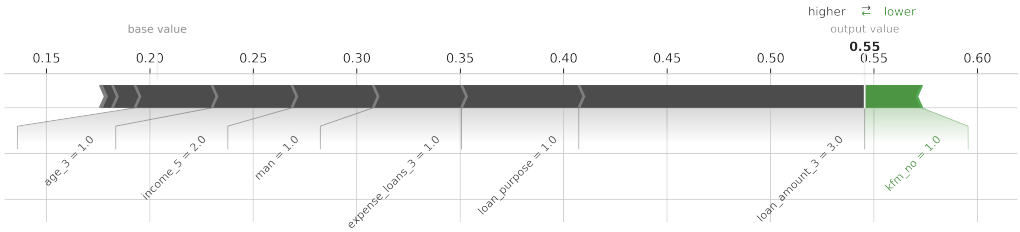

Figure 5.7: Individual explanation of MLP on a "bad" customer prediction

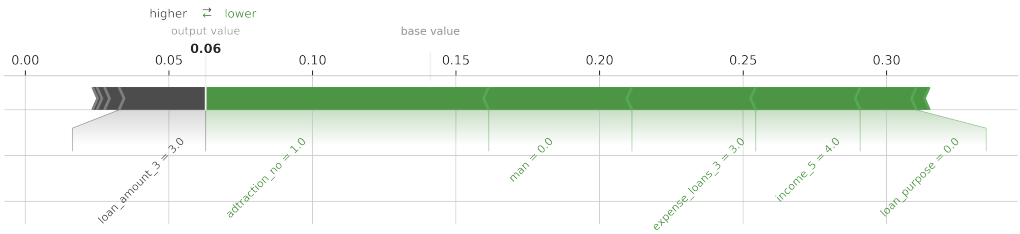

Figure 5.8: Individual explanation of LOG on a "good" customer prediction

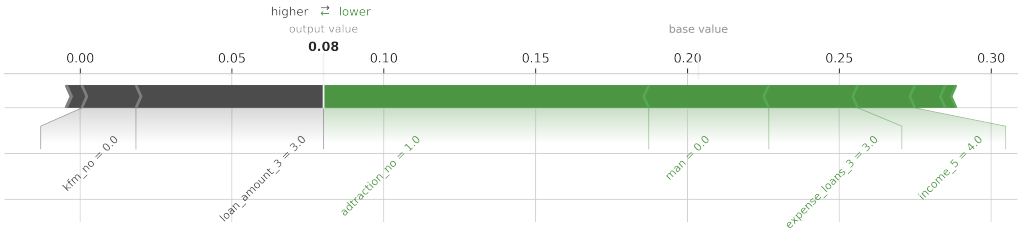

Figure 5.9: Individual explanation of MLP on a "good" customer prediction

The individual explanations provide an interpretable visualization of how the models made their classification. As expected from Figure 5.4 and 5.5, the logistic regression model and the neural network performed quite similarly.

5.4.4 Explaining individual predictions through LIME

In Figure 5.10 and 5.11, individual predictions are explained using the LIME technique for correctly classified "good" and "bad" predictions, where each bar represents the importance of the corresponding input feature.

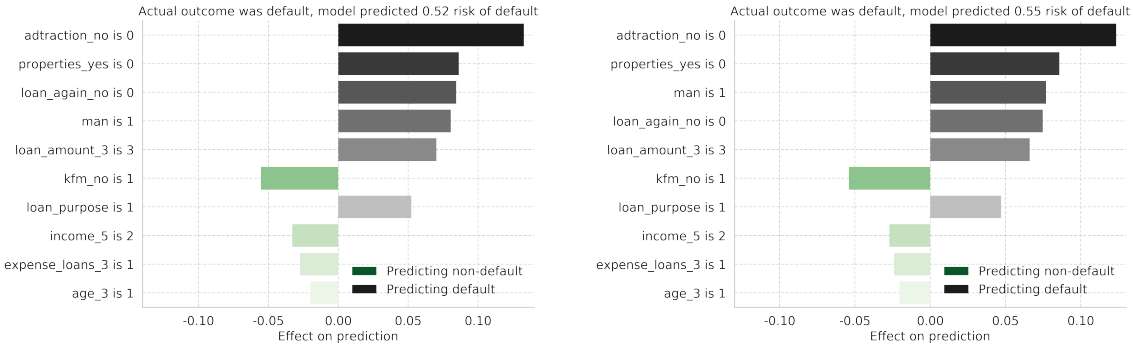

Figure 5.10: Individual explanation of predictions by logistic regression and neural network using LIME on a "bad" customer prediction

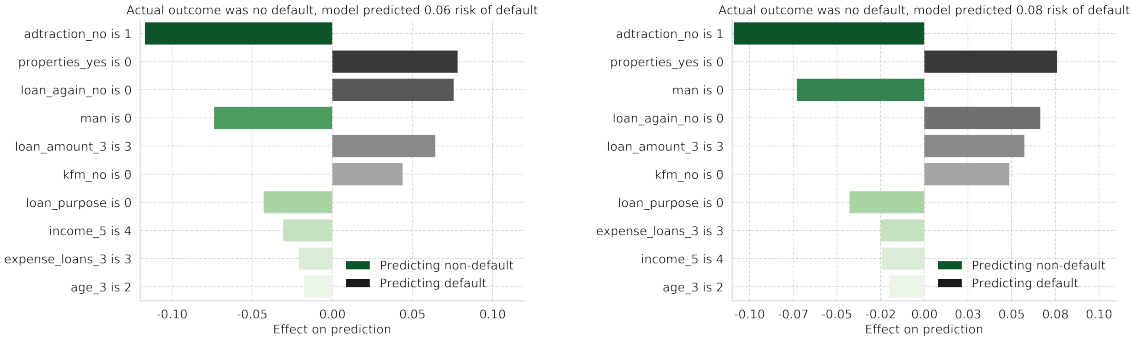

Figure 5.11: Individual explanation of predictions by logistic regression and neural network using LIME on a "good" customer prediction

Much like SHAP, the LIME explanation technique provides a tool for interpreting and visualizing individual predictions.

Chapter 6
Discussion

6.1 Model prediction performance

By comparing the overall average performance in Section 5.3.1, there is no clear superior model. Logistic regression, support vector machine, neural network and the soft voting ensemble seem to perform quite equal on all intervals. Hence, it is hard to draw any conclusions from these results alone since there is a difference of about 1-3% in precision between the models. For a certain cutoff level one model achieves the highest precision, and for another cutoff level the same model was not the best performing, which is visualized in Figure 5.1. Random forest was not able to achieve the same average precision as the other models on recall level 20-35%, which could be explained by that it required strong regularization in order not to overfit completely. The k-Nearest Neighbors and decision tree models were not performing well despite various attempts and were therefore rejected on an early stage.

A key insight was that the most important input feature without doubt was the loan amount that the loan applicant had applied for. For the largest loan amounts (in the interval 21-25kSEK) the most successful models had a precision of >54% at recall level 20-35% and a precision of >51% at recall level 35-50% (see Table 5.3). For mid-sized loans (in the interval 4-20kSEK), the most successful models had a precision of >51% at recall level 20-35% and a precision of >37% at recall level 35-50% (see Table 5.4). For the smallest loan amounts however, the precision were about 18-21%, which is almost equal to random guessing (see Table 5.5). It was expected that it would be easier to classify larger loans because of the higher default frequency, but perhaps not by this magnitude. This suggests that the models perhaps should only be used to classify loan applications on loans between 4-25kSEK, or to achieve even higher precision, on loans between 21-25kSEK. As discussed in Section 4.2.3, the loans in the interval 21-25kSEK accounts for 53% of the Company's total lending volume, and 62% of the total defaulted lending volume. Given the volume of this group in terms of capital and that it has a default frequency of almost 25%, this subset of customers is the most crucial to classify accurately. Why the models fail to classify the smallest loans could be because of the low default frequency of $\sim 9\%$ compared with the data set overall of 17.8%.

6.2 Model complexity and transparency

In Section 5.4, prediction explanations using SHAP values and LIME was illustrated for the neural network and the logistic regression model. These models were chosen in order to illustrate the possibility of explaining a black-box model's predictions as well as comparing the explanation with a white-box model's such as logistic regression. Since SHAP and LIME use two completely different approaches in estimating an input feature's importance, it was also expected that the explanations would slightly differ as well. The two methods' output showed quite similar results, and both are able to explain an individual prediction quite illustrative and logical. Input features that showed a higher default risk during the feature engineering process were also showing a higher effect on the

classification decision according to LIME and SHAP. To get the insight that the neural network uses input features logically to classify a data sample instead of using non-logical relationships such as sample noise increases the model reliability. This means that when using the model, the credit analyst can refer to the results of the explanation to understand the underlying model decision for an individual prediction using either LIME or SHAP.

A clear advantage with the SHAP approach is that it can provide an overall explanation for the entire data set, which is shown in Figure 5.4. The overall explanation seems to be consistent with the individual predictions that were performed, which increases the reliability of the SHAP approach. For LIME on the other hand, there is no currently available implementation such that it can give an overall explanation of the whole data set. Not knowing the explanation over the entire data set means that it is unknown whether the explanation is consistent over all predictions. Therefore, a more reliable approach would be to use SHAP to get both individual explanations as well as an overall explanation.

The fact that increased complexity did not improve model performance was not a surprise. Given the relatively small data set of 4052 samples to both train and test the models, a simple logistic regression model should theoretically perform almost equal to e.g. a strongly regularized neural network. Also, when training the support vector machine, the non-linear classifiers showed no improvements, which would strengthen this assumption. Furthermore, the numerical input features were re-coded in such a way that it would suit logistic regression by constructing simple generalized dummy variables. If the input features were used without any re-coding, more complex models might fare better than logistic regression.

By using SHAP or LIME, the issue with black-box models' inability to provide a prediction explanation is somewhat dealt with. Nonetheless, when choosing between two models that perform equal, a reasonable choice would be to use the one with the lowest complexity and highest transparency. This would suggest that the logistic regression model is superior to e.g. the neural network or the soft voting ensemble, since it's easier to update and adjust and is easier to comprehend when using it. However, the more complex models should not be rejected in the long-term since they may improve performance when more data samples and input features become available in the future.

6.3 Data insufficiency and stability

When evaluating model performance, the results varied significantly depending on how the train/test split was made. For one specific split, one model was superior whereas for another split another one performed better. This was an issue for all of the models, which could be explained by the limited amount of data since all models were affected. To avoid the problem with instability, in Section 5.3.1 the model performance was instead cross-validated on an average of 18 different splits where the models were re-trained on each split to get the model that performed the best over the entire data set. These results showed that logistic regression, neural network and the soft voting ensemble performed similarly, with support vector machine and random forest showing slightly worse performance, especially for large-sized loans. Using any of the classifiers, one should however be aware that the final precision may deviate quite significantly, although on acceptable levels (standard deviation of approximately 3.5% for the average precision scores).

The models were also tested to ensure that they were stable over time. This was done by splitting the data chronologically by training the models on the "oldest" 80% of the loan applications and testing the performance on the "newest" 20%. By studying the results in Section 5.3.3, the models do not seem to lose more than limited predictive power over time, i.e. time dependency or seasonality seems not to be that much of an issue. Comparing to the cross-validated performance tests in Section 5.3.1, the "newest" 20% of the gathered data seem to be quite representative of the data set as a whole. At least the model does not seem to have trouble predicting defaults in

the newer data relative to predicting defaults in the whole data set, which is a satisfying result given that the model will be used on future data. This is however not a guarantee that the model will, if implemented, perform just as good on new data, but it is a moderate indication that the performance will not sharply decline over time.

The issue with lack of data samples was examined in Section 5.3.2 where the models performed significantly worse when training on only 7% and 35% of the available data as opposed to the original training k-folds of 70%. There is a somewhat log-linear relationship between data volume and performance which would suggest that the models will improve even further when more data becomes available in the future, even though this is not a guarantee.

As seen during the feature engineering process in Section 4.2, the only input features available are to be seen as relatively weak predictors. Even though there were many variables that in combination yielded a stronger discriminatory power, the theoretical precision and recall that can be achieved is somewhat limited. A precision of 50% could be interpreted as a poor performance compared to e.g. classification of flower species, where features are available that enables a precision well above 90%. However, given the limited amount of data and weak predictors and the seemingly difficult classification problem of scorecard modelling, the preferred way believed to improve the models is a combination of 1) gather more data samples, and 2) find more input features that could have a higher discriminatory power. Of course, extended fine-tuning of the models could be performed, but is expected to have only marginal effect on performance.

6.4 Conclusion

Given that four models with varying complexity performed equally well in terms of prediction performance and stability, there is no additional value in increased complexity, rather the opposite. Thus, the logistic regression model which is the simplest of the well-performing models should be proposed for further use. However, more sophisticated models such as the neural network should not be ruled out for future use, since it has shown superior performance in complex decision processes where large data sets can be used for training. This is nonetheless ahead in time and should be regarded as a long-term experimental activity.

Since the Company has no current statistical scoring model implemented long enough that can be benchmarked against, it is hard to determine whether the results actually offer any improvements in terms of increased profitability. By using a machine learning classifier however, the Company has the potential to reject approximately 20-35% of its current "bad" customers to the cost of rejecting one "good" customer for each "bad" (given a $\sim 50\%$ precision as shown in Section 5.1). This 50/50 trade-off seems reasonable in a profitability perspective, since one defaulting customer will by all means be a higher cost than the lost revenue of rejecting a non-defaulting customer, given the Company's loan interest rates of about 30%. Thus, even without an existing model to benchmark against, it can be concluded that it would be financially favorable for the Company to apply a model to classify loan applications.

Future performance cannot be guaranteed in terms of stability and time-dependency, despite promising indications showed in the results. If a model were to be implemented, the recommendation is to use it with caution. The model could initially be tested and evaluated on a small portion of the loan applications before using it on a full scale given that the results are satisfactory. When more data samples and/or input features becomes available, the models should regularly be re-trained and evaluated to ensure both precision and stability. Optimally, a high precision/ low recall configuration should be used in order not to jeopardize revenues by incorrectly rejecting too many "good" customers. Profitability-wise, incorrectly classifying "bad" customers as "good" is not considered an introduced business risk since the Company would have accepted these customers anyways if no model were used. Incorrectly classifying too many "good" customers as "bad" on the other hand is more problematic, since it potentially could cause falling revenues without lowering the default rates enough to compensate. The risk of having a configuration with high recall is that the model might show poor precision, which could be costly in terms of losing "good" customers. Anyhow, machine learning classification shows great potential in classifying high-risk consumer credits and should definitely be considered to be included in the Company's application approval process.

Chapter 7
Further work

7.1 Profitability analysis

With a reliable classification model at hand, the next step is to calculate the profitability of a loan for a given customer in order to set the optimal trade-off between precision and recall. This can be seen as a fairly complex analysis that involves calculating revenues (such as interest rate, reminder charges etc.) and costs (such as marketing costs, personnel, financial expenses etc.) for each loan based on its loan size, as well as determining the recovery rate for a defaulted loan (i.e. how many percent of the principal amount of a default is an actual loss). This is beyond the scope of this report but is a central part in order to configure the model efficiently, since increased profitability together with regulatory purposes would be the ultimate goal of using a scoring model.

7.2 Gathering of new input features- Extended feature engineering

As discussed, perhaps the largest area of model improvement lies in finding input features with stronger predictive power. One action to take is for the Company to gather vast amounts of information both retrospectively and prospectively and evaluate if there are any significant findings in terms of discriminatory power. Also, there are potentially combined variables that can be created out of existing ones that could improve the model performance further.

Chapter 8

References

[Bishop 2006] Bishop, Christopher M.: *Pattern Recognition and Machine Learning*. Cambridge CB3 0FB, U.K : Microsoft Research Ltd, 2006. – Available at: https://bit.ly/2dfDYWs

[Bolton 2009] Bolton, Christine: *Logistic regression and its application in credit scoring*. Pretoria, South Africa : University of Pretoria, 2009. – Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1024.2660&rep=rep1&type=pdf

[Bousquet and Elisseeff 2002] Bousquet, Olivier ; Elisseeff, André: Stability and Generalization. In: *Journal of Machine Learning Research* 2 (2002), S. 499–526. – Available at: http://www.jmlr.org/papers/volume2/bousquet02a/bousquet02a.pdf

[Breiman 2001] Breiman, Leo: *Random Forests*. Berkeley, CA 94720 : Statistics Department, University of California, 2001. – Available at: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[Buitinck et al. 2013] Buitinck, Lars ; Louppe, Gilles ; Blondel, Mathieu ; Pedregosa, Fabian ; Mueller, Andreas ; Grisel, Olivier ; Niculae, Vlad ; Prettenhofer, Peter ; Gramfort, Alexandre ; Grobler, Jaques ; Layton, Robert ; VanderPlas, Jake ; Joly, Arnaud ; Holt, Brian ; Varoquaux, Gaël: API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, S. 108–122

[Cawley and Talbot 2010] Cawley, Gavin C. ; Talbot, Nicola L. C.: On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. (2010). – Available at: http://www.jmlr.org/papers/volume11/cawley10a/cawley10a.pdf

[Chawla et al. 2002] Chawla, Nitesh V. ; Bowyer, Kevin W. ; O.Hall, Lawrence ; Kegelmeyer, W.Philip: SMOTE: Synthetic Minority Over-sampling Technique. In: *Journal of Artificial Intelligence Research* 16 (2002), S. 321–357. – Available at: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume16/chawla02a-html/chawla2002.html

[Engelmann and Rauhmeier 2011] Engelmann, Bernd ; Rauhmeier, Robert: *The Basel II Risk Parameters- Second edition* . Springer-Verlag Berlin Heidelberg, 2011. – Available at: http://www.hkfrm.org/resources/Risk_Parameters.pdf

[Fawcett 2006] Fawcett, Tom: An introduction to ROC analysis. In: *Pattern Recognition Letters* 27 (2006), Nr. 8, S. 861–874. – Available at: https://doi.org/10.1016/j.patrec.2005.10.010

[Goodfellow et al. 2016] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – Available at: http://www.deeplearningbook.org

[He et al. 2008] He, Haibo ; Bai, Yang ; Garcia, Edwardo A. ; Li, Shutao: *ADASYN: Synthetic Sampling Approach for Imbalanced Learning*. International Joint Conference on Neural Network,

Universidad de Granada, 2008. – Available at: https://sci2s.ugr.es/keel/pdf/algorithm/congreso/2008-He-ieee.pdf

[Kohavi 1995] Kohavi, Ron: *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. Morgan Kaufmann, 1995. – 1137–1143 S. – Available at: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.529

[Lundberg and Lee 2017] Lundberg, Scott M. ; Lee, Su-In: A Unified Approach to Interpreting Model Predictions. In: *Part of: Advances in Neural Information Processing Systems 30 (NIPS)* (2017). – Available at: https://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[MIT-OpenCourseWare retrieved 2019] MIT-OpenCourseWare: *Lecture on Support Vector Machines*. retrieved 2019. – Available at: http://courses.csail.mit.edu/6.034f/ai3/SVM.pdf

[Murphy 2006] Murphy, Kevin P.: *Machine Learning- A Probabilistic Perspective*. Cambridge, Massachusetts : The MIT Press, Massachusetts Institute of Technology, 2006

[Pedregosa et al. 2009–2018] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ; Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E.: Scikit-learn: Machine Learning in Python,v0.20.3. In: *Journal of Machine Learning Research* 12 (2009-2018)

[Ribeiro et al. 2016] Ribeiro, Marco T. ; Singh, Sameer ; Guestrin, Carlos: *"Why should I trust you?" Explaining the Predictions of Any Classifier*. Seattle, WA 98105, USA : University of Seattle, 2016. – Available at: https://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf

[Ruder 2017] Ruder, Sebastian: *An overview of gradient descent optimization algorithms*. Insight Centre for Data Analytics, NUI Galway, 2017. – Available at: https://arxiv.org/pdf/1609.04747.pdf

[Russell and Norvig 2010] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach, 3rd ed.* Upper Saddle River, New Jersey 07458 : Pearson Education Inc, 2010

[Srivastava et al. 2014] Srivastava, Nitish ; Hinton, Geoffrey ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan: Dropout: A simple way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* 15 (2014), S. 1929–1958. – Available at: http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[Swedish Code Of Statutes 2018] Swedish Code Of Statutes: *SFS 2018:478 Lag om ändring i konsumentkreditlagen (2010:1846)*. 2018. – Available at: https://www.lagboken.se/Lagboken/sfs/sfs/2018/400-499/d_3243282-sfs-2018_478-lag-om-andring-i-konsumentkreditlagen-2010_1846

Appendix A

Input Features

Variable	Type	Description
gender	Binary	man/woman
kfm	Binary	history from Kronofogdemyndigheten
loan again	Binary	Intention to loan again
property owner	Binary	owning a property yes/no
recurring customer	Binary	recurring customer yes/no
age	Numerical	age (years)
expense children	Numerical	childcare expenses (kSEK)
expense general	Numerical	other expenses (kSEK)
expense household loans	Numerical	household's loan expenses (kSEK)
expense loans	Numerical	individual loan expenses (kSEK)
expense rent	Numerical	individual rent expenses (kSEK)
expense travels	Numerical	travel expenses (commuting etc)
income household	Numerical	household's income (kSEK)
income individual	Numerical	individual income (kSEK)
loan amount	Numerical	application's credit limit (kSEK)
nbr of adults	Numerical	nbr of adults in household
nbr of kids	Numerical	nbr of kids in household
property value	Numerical	value of property (kSEK)
surplus household	Numerical	household's surplus (kSEK)
surplus individual	Numerical	individual surplus (kSEK)
time	Numerical	time of day the application was made (h)
civil status	Qualitative	civil status (4 categories)
loan purpose	Qualitative	reason for loan application (6 categories)
mkting channel	Qualitative	how applicant reached the website
weekday	Qualitative	weekday of application (Mon-Sun)

Table A.1: All available input features before any re-coding

Variable	Type
loan amount $<$ 4000	Binary
loan amount $>$ 20001	Binary
income $<$ 15001	Binary
income $>$ 25001	Binary
age $<$ 25	Binary
age $>$ 56	Binary
expense loans $<$ 2000	Binary
expense loans $>$ 7001	Binary
man	Binary
kfm no	Binary
properties yes	Binary
loan purpose	Binary
loan again no	Binary
adtraction no	Binary
married yes	Binary

Table A.2: Feature Subset 1, with all variables as dummies

Variable name	Type	Values
man yes	Binary	0 or 1
income 5	Categorized numerical	(0-15|16-18|19-22|23-27|27.5-)
age 5	Categorized	(18-25| 26-30|31-45|46-54|55-)
loan amount 3	Categorized	(0-4|5-20|21-25)
expense loans 3	Categorized	(0-3|4-6|7-)
kfm no	Binary	0 or 1
loan purpose	Binary	0 or 1
loan again no	Binary	0 or 1
adtraction no	Binary	0 or 1
properties yes	Binary	0 or 1
married yes	Binary	0 or 1

Table A.3: Feature Subset 2, with numerical variables re-coded into 5 and 3 ranked categories respectively

Variable	Type
loan amount	Numerical
income	Numerical
age	Numerical
expense loans	Numerical
man	Binary
kfm no	Binary
properties yes	Binary
loan purpose	Binary
loan again no	Binary
adtraction no	Binary
married yes	Binary

Table A.4: Feature subset 3, with numerical variables not being re-coded as in the case in Feature Subset 2

Variable name	Type	Values
loan amount $<$ 4000	Binary	0 or 1
loan amount $>$ 20001	Binary	0 or 1
income 5	Categorized numerical	(0-15|16-18|19-22|23-27|27.5-)
age $<$ 25	Binary	0 or 1
age $>$ 56	Binary	0 or 1
expense loans $<$ 2000	Binary	0 or 1
expense loans $>$ 7001	Binary	0 or 1
man	Binary	0 or 1
kfm no	Binary	0 or 1
properties yes	Binary	0 or 1
loan purpose	Binary	0 or 1
loan again no	Binary	0 or 1
adtraction no	Binary	0 or 1

Table A.5: Feature Subset 4 - the one used in the final models, with combined binary and re-coded numerical variables

Appendix B
Supplementary results

B.1 Permutation importance for random forest, support vector classifier and soft voting ensemble

Weight	Feature
0.1105 ± 0.0524	loan_amount_3
0.0481 ± 0.0216	adtraction_no
0.0333 ± 0.0301	properties_yes
0.0308 ± 0.0361	man
0.0272 ± 0.0419	expense_loans_3
0.0249 ± 0.0304	loan_purpose
0.0205 ± 0.0409	income_5
0.0173 ± 0.0245	loan_again_no
0.0133 ± 0.0255	age_3
0.0060 ± 0.0167	kfm_no

Weight	Feature
0.1396 ± 0.0139	loan_amount_3
0.0769 ± 0.0091	expense_loans_3
0.0673 ± 0.0185	loan_purpose
0.0665 ± 0.0208	income_5
0.0613 ± 0.0175	man
0.0559 ± 0.0140	adtraction_no
0.0507 ± 0.0072	age_3
0.0245 ± 0.0093	loan_again_no
0.0245 ± 0.0140	properties_yes
0.0197 ± 0.0054	kfm_no

Figure B.1: Permutation importance for random forest on test set (left) and train set (right)

Weight	Feature
0.1266 ± 0.0522	loan_amount_3
0.0416 ± 0.0185	adtraction_no
0.0307 ± 0.0457	man
0.0272 ± 0.0213	properties_yes
0.0210 ± 0.0344	loan_again_no
0.0198 ± 0.0214	expense_loans_3
0.0197 ± 0.0153	loan_purpose
0.0168 ± 0.0314	age_3
0.0080 ± 0.0380	income_5
0.0045 ± 0.0432	kfm_no

Weight	Feature
0.0976 ± 0.0219	loan_amount_3
0.0390 ± 0.0159	adtraction_no
0.0320 ± 0.0155	expense_loans_3
0.0299 ± 0.0156	man
0.0283 ± 0.0150	income_5
0.0189 ± 0.0115	age_3
0.0121 ± 0.0165	properties_yes
0.0077 ± 0.0102	loan_again_no
0.0061 ± 0.0059	loan_purpose
0.0039 ± 0.0116	kfm_no

Figure B.2: Permutation importance for support vector machine on test set (left) and train set (right)

Weight	Feature
0.1330 ± 0.0489	loan_amount_3
0.0457 ± 0.0182	adtraction_no
0.0308 ± 0.0414	man
0.0294 ± 0.0286	loan_purpose
0.0278 ± 0.0223	properties_yes
0.0228 ± 0.0378	income_5
0.0208 ± 0.0290	loan_again_no
0.0199 ± 0.0215	expense_loans_3
0.0181 ± 0.0242	age_3
0.0129 ± 0.0420	kfm_no

Weight	Feature
0.1067 ± 0.0225	loan_amount_3
0.0445 ± 0.0178	adtraction_no
0.0380 ± 0.0153	income_5
0.0352 ± 0.0166	expense_loans_3
0.0347 ± 0.0138	man
0.0189 ± 0.0077	age_3
0.0188 ± 0.0147	properties_yes
0.0183 ± 0.0121	loan_purpose
0.0110 ± 0.0113	loan_again_no
0.0043 ± 0.0106	kfm_no

Figure B.3: Permutation importance for soft voting ensemble on test set (left) and train set (right)

B.2 Explaining overall predictions through SHAP for random forest, support vector classifier and soft voting ensemble

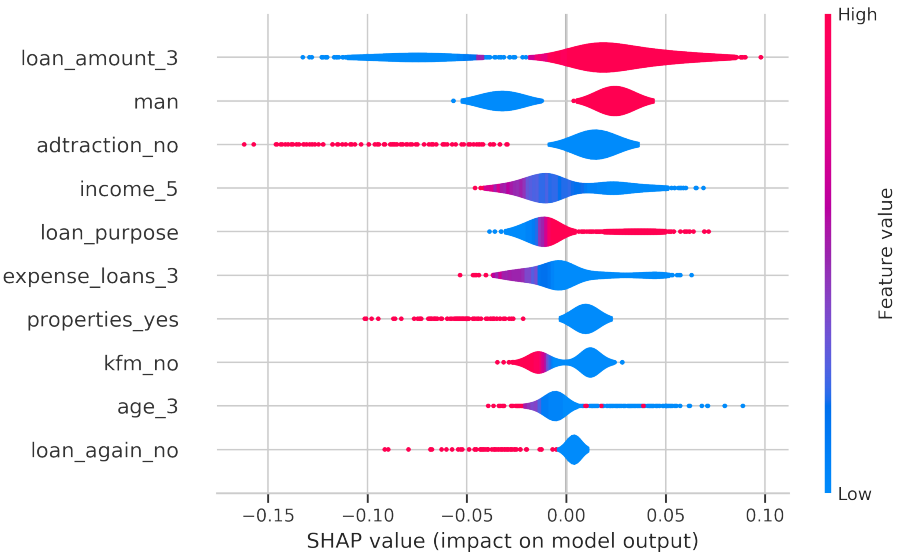

Figure B.4: Overall effect on predictions by random forest per feature by SHAP value

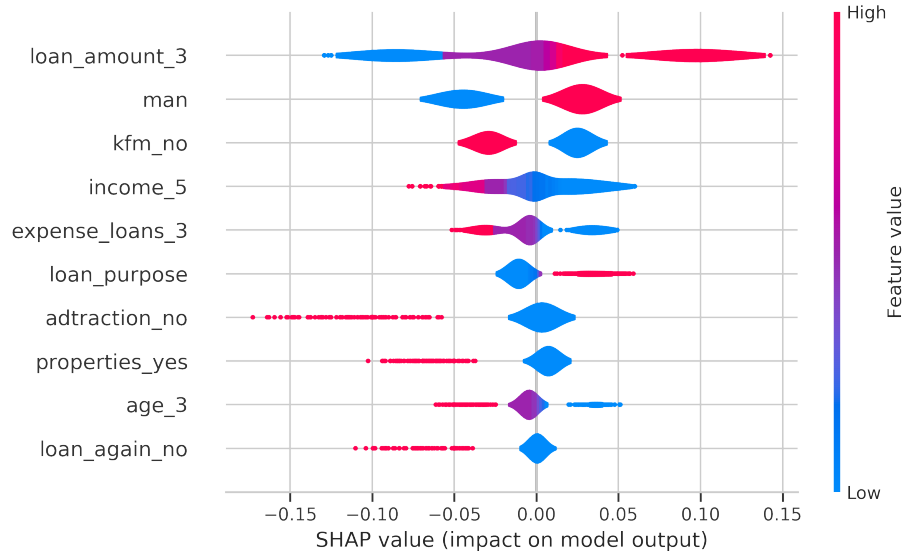

Figure B.5: Overall effect on predictions by support vector machine per feature by SHAP value

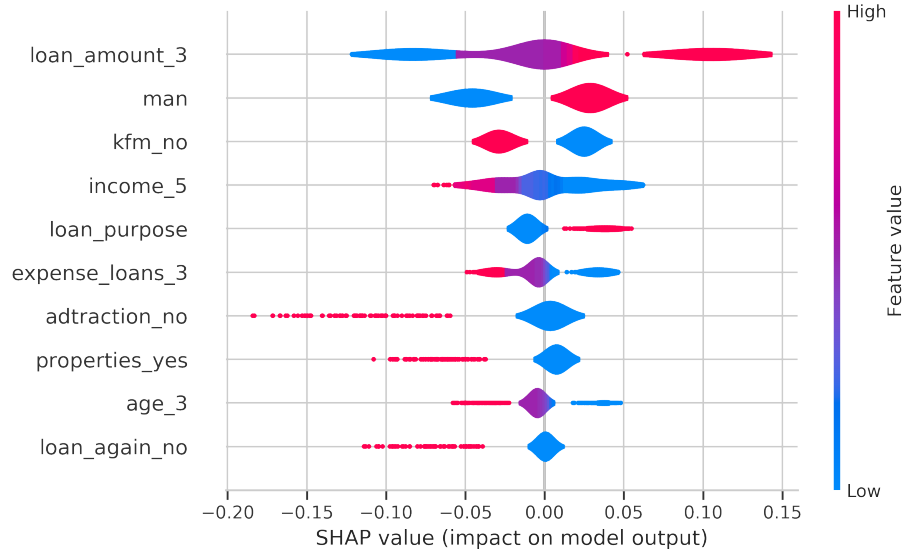

Figure B.6: Overall effect on predictions by soft voting ensemble per feature by SHAP value

B.3 Explaining individual predictions through SHAP for random forest, support vector classifier and soft voting ensemble

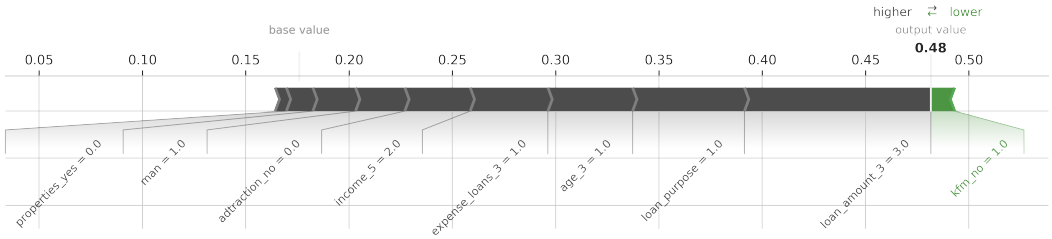

Figure B.7: Individual explanation of RFO on a "bad" customer prediction

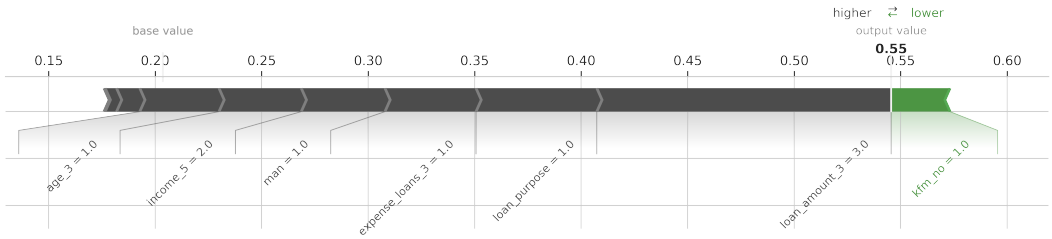

Figure B.8: Individual explanation of SVC on a "bad" customer prediction

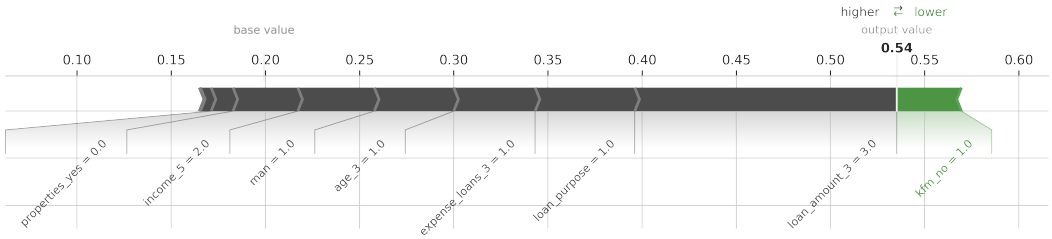

Figure B.9: Individual explanation of SVO on a "bad" customer prediction

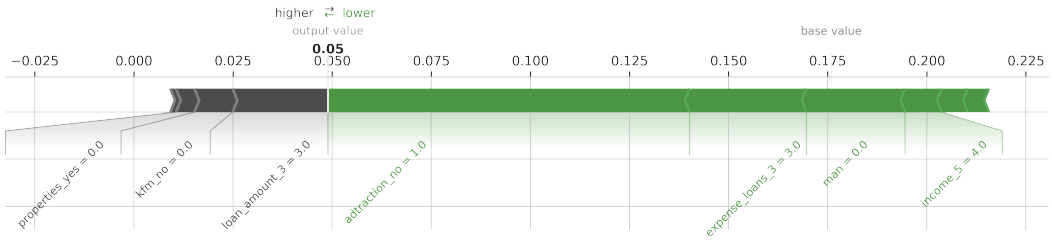

Figure B.10: Individual explanation of RFO on a "good" customer prediction

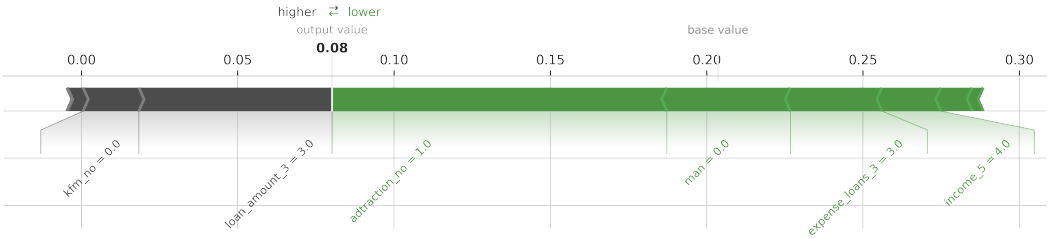

Figure B.11: Individual explanation of SVC on a "good" customer prediction

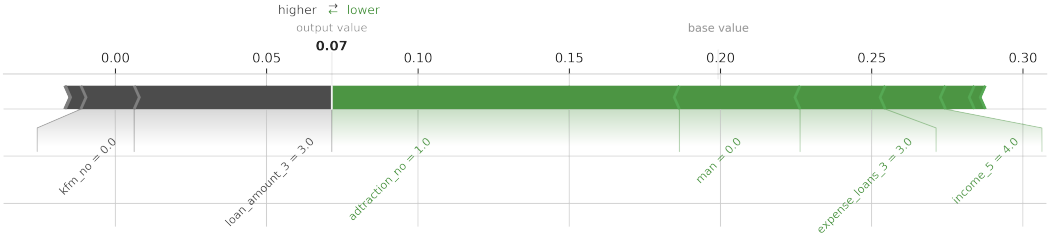

Figure B.12: Individual explanation of SVO on a "good" customer prediction

B.4 Explaining individual predictions through LIME for random forest, support vector classifier and soft voting ensemble

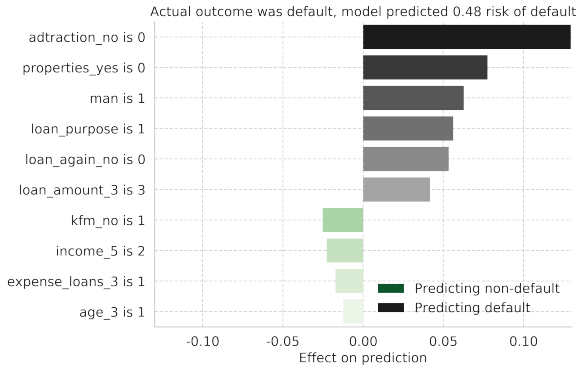

Figure B.13: Individual explanation of prediction by random forest using LIME on an actual "bad" customer

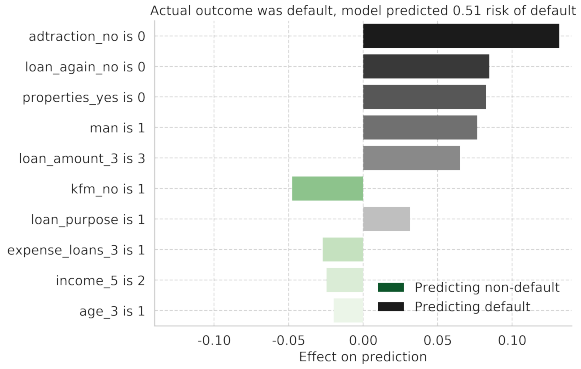

Figure B.14: Individual explanation of prediction by support vector classifier using LIME on an actual "bad" customer

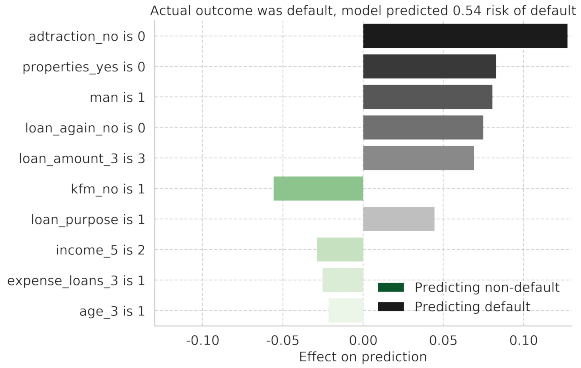

Figure B.15: Individual explanation of prediction by soft voting ensemble using LIME on an actual "bad" customer

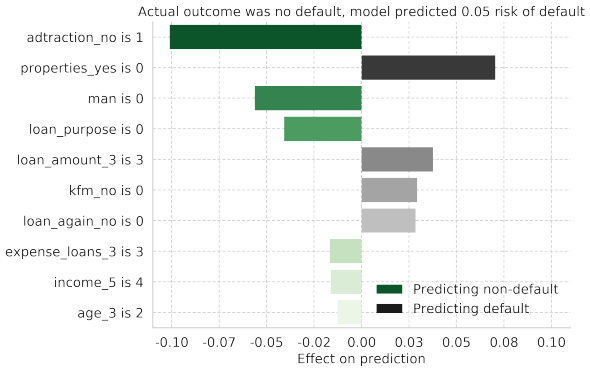

Figure B.16: Individual explanation of prediction by random forest using LIME on an actual "good" customer

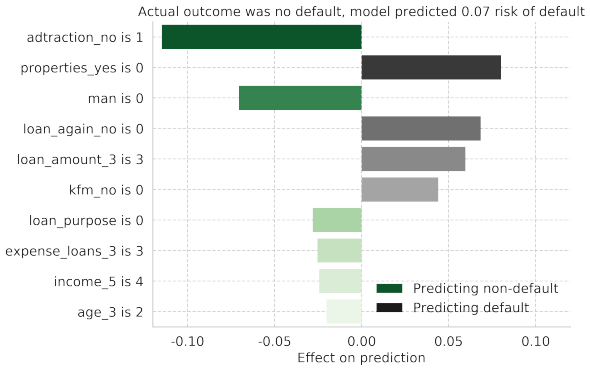

Figure B.17: Individual explanation of prediction by support vector classifier using LIME on an actual "good" customer

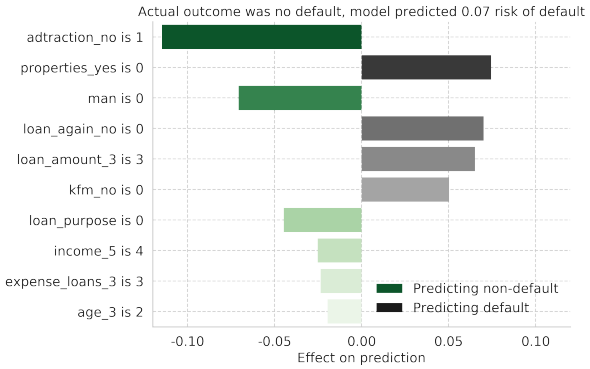

Figure B.18: Individual explanation of prediction by soft voting ensemble using LIME on an actual "good" customer